# Deep Neural Networks for Large-Scale Cytoarchitectonic Mapping of the Human Brain

Inaugural-Dissertation

zur Erlangung des Doktorgrades
der Mathematisch-Naturwissenschaftlichen Fakultät
der Heinrich-Heine-Universität Düsseldorf

vorgelegt von

**Christian Schiffer**
aus Geilenkirchen

Düsseldorf, März 2022

aus dem Institut für Informatik
der Heinrich-Heine-Universität Düsseldorf

Gedruckt mit der Genehmigung der
Mathematisch-Naturwissenschaftlichen Fakultät der
Heinrich-Heine-Universität Düsseldorf

*Berichterstatter*:

- Prof. Dr. med. Katrin Amunts
- Prof. Dr. Stefan Harmeling

*Tag der mündlichen Prüfung*:
17.11.2022


# Abstract

The analysis of microstructurally distinct cytoarchitectonic areas in the human brain provides the foundation to associate functional, physiological, genetic, molecular, and connectivity data with anatomically well-defined entities. Cytoarchitecture encompasses characteristic properties of neuronal cell distributions, including their shape, size, and spatial organization. High-resolution microscopic scans of cell-body stained histological brain sections enable the detailed analysis of these cytoarchitectonic properties, and thereby the brain's parcellation into structurally defined areas. The high inter-individual variability between brains necessitates the analysis of multiple brains to obtain a general picture of the human cytoarchitectonic organization. Modern high-throughput scanners enable the acquisition of microscopic image data on a large scale. However, established cytoarchitecture analysis methods are infeasible to handle the steadily increasing volume of data. This motivates the development of methods for the automated classification of cytoarchitectonic brain areas. Previous works on automated cytoarchitecture classification demonstrated the potential of deep learning methods to address this challenging task.

This work addresses automated cytoarchitectonic brain mapping at large scale. It introduces a deep learning method for interactive classification of individual brain areas across large series of histological brain sections. The method exploits the limited local variability of individual brain areas and requires minimal annotations. It integrates well with existing brain mapping workflows and provides the first practical method to support cytoarchitectonic mapping in large series of sections. Results of the presented workflow provide the foundations for creating 3D reconstructions of individual brain areas at previously unachieved spatial resolution.

The developed workflow focuses on the interactive application of deep learning for supporting cytoarchitectonic mapping. As a step towards fully automated brain mapping at large scale, this work further explores deep learning methods for classifying many different brain areas in multiple brain samples. It introduces a supervised contrastive learning method that learns to extract cytoarchitectonic features from large microscopic image datasets. Comprehensive evaluations demonstrate that learned features capture meaningful neuroanatomical properties and enable the accurate prediction of different cytoarchitectonic areas.

Finally, this work introduces a framework for cytoarchitectonic mapping using graph neural networks. It models the task as a node classification problem in a graph, enabling efficient integration of local cytoarchitectonic features with topological and contextual information. The approach takes inspiration from existing brain mapping workflows and achieves significantly improved classification performance.

# Acknowledgements

The studies presented in this doctoral thesis were conducted in the Big Data Analytics (BDA) group and the Helmholtz AI research group "Artificial intelligence for decoding human brain organization", both led by Prof. Timo Dickscheid at the Institute of Neuroscience and Medicine (INM-1) led by Prof. Katrin Amunts at Forschungszentrum Jülich (FZJ). It was submitted to the Heinrich-Heine-University Düsseldorf (HHU) to obtain a doctoral degree in computer science.

I want to thank Prof. Katrin Amunts (INM-1, FZJ), who supported this work as my thesis advisor and provided invaluable advice on the neuroscientific aspects of this interdisciplinary thesis. She always found the time to answer my questions on neuroscientific topics and to provide valuable comments on drafts of several papers and this thesis. I am extremely grateful that she gave me the opportunity to attend various scientific conferences, where I could present my work to the scientific community, and for the opportunity to collaborate with many great researchers.

I am very grateful for the support of my second thesis advisor Prof. Stefan Harmeling (HHU). The discussions with him opened up new perspectives on the project, inspired new ideas, and contributed greatly to the project's success. I appreciate his feedback on drafts of this thesis and several papers, which greatly contributed to the development of my writing skills.

I would like to thank Prof. Timo Dickscheid (INM-1, FZJ), who gave me the opportunity to work on this exciting project. His continuous support and efforts to provide a working environment that allowed me to pursue and develop my own ideas have greatly contributed to the success of this project. He always found time to discuss new ideas, provided valuable advice, and motivated me to expand my knowledge of image processing, machine learning, brain anatomy, and many other fields. His valuable comments on drafts for several papers, proposals, and this thesis allowed me to refine my writing skills and formulate my ideas clearly and concisely.

I would like to express my deepest appreciation to Dr. Hannah Spitzer, who laid the foundation for this thesis with her pioneering work on automated cytoarchitectonic brain mapping. The discussions with her helped me to greatly extend my knowledge of deep learning, image processing, neuroanatomy, and cytoarchitectonic mapping. I greatly appreciate the time and effort she invested in introducing me to the details and challenges of automated brain mapping, without which this thesis would not have been possible.

Special thanks go to Kai Kiwitz (Cécile and Oskar Vogt Institute for Brain Research, HHU), who never grew tired of answering my many questions regarding the anatomy of the human brain and cytoarchitectonic brain mapping. His invaluable feedback on the results of the interactive mapping method (Chapter 5) greatly contributed to the development of the method and ultimately to the creation of 3D cytoarchitectonic maps at previously unachieved resolution. I would like to thank Andrea Brandstetter, Dr. Sebastian Bludau, and Dr. Nicola Palomero-Gallagher (INM-1, FZJ), who helped me to better understand the challenges of cytoarchitectonic brain mapping and thereby inspired the development of better automated brain mapping methods. My sincere thanks go to Hartmut Mohlberg (INM-1, FZJ) for providing annotations of cytoarchitectonic brain areas and volumetric transformations that were used in this thesis.

I would like to thank the members of the ACELab at the Montréal Neurological Institute, McGill University (MNI) in Canada, led by Prof. Alan C. Evans, with who I had the pleasure to collaborate through the Helmholtz International Laboratory HIBALL. In particular, I would like to thank Claude Lepage for assisting me with the 3D reconstruction of BigBrain. I want to thank the team of the Jülich Supercomputing Centre (JSC) for providing an outstanding computing infrastructure and exceptional support. In particular, I want to thank Sebastian Lührs (JSC, FZJ) for his assistance regarding the HPST storage system, which played a crucial role in many of the conducted experiments. I would also like to thank Fahad Khalid (JSC, FZJ) and Dr. Andreas Herten (JSC, FZJ) for their support with the supercomputing systems and for giving me the opportunity to discuss and present my project with members of the HPC community.

I would like to thank all of my colleagues at the INM-1 for their support on countless occasions and for the welcoming and pleasant working atmosphere. In particular, I would like to thank Sarah Oliveira, Marcel Huysegoms, Eric Upschulte, Esteban Alejandro Vaca Cerda, Alexander Oberstraß, and Ahmed Nebli for the many enlightening discussions and valuable suggestions. I also want to thank all participants of the regular after-lunch-lake-round, which has more than once been the source of exciting new ideas. Special thanks goes to Dr. Susanne Wenzel for keeping everything together.

Last but not least, I owe my deepest gratitude to my family for their everlasting support and encouragement. My deepest appreciation goes to my parents, Jutta and Dieter Schiffer, my brother Stefan Schiffer, and to Carina Päßler, who keeps everything going. Thank you.

## Funding information

The project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 720270 (HBP SGA1) and Grant Agreement No. 785907 (HBP SGA2). It was supported by the EBRAINS research infrastructure, funded from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreement No. 945539 (HBP SGA3). It received funding from the Helmholtz Association's Initiative and Networking Fund through the Helmholtz International BigBrain Analytics and Learning Laboratory (HIBALL) under the Helmholtz International Lab grant agreement InterLabs-0015. Computing time was granted through JARA on the supercomputers JURECA and JURECA-DC at JSC.

# Contents

# 1 Introduction

Understanding the human brain is one of humanity's greatest endeavors. Studying its functional and structural organization is the key to a deeper understanding of the mechanisms that enable us to perform complex cognitive tasks. Extending our knowledge on the structure of healthy brains will further enable us to better understand the effects of neurodegenerative diseases like Parkinson's and Alzheimer's disease.

Brain atlases are essential tools used in modern neuroscience (Amunts et al., 2015). Similar to how geographical atlases consolidate information on various geographic features and political boundaries, brain atlases integrate information on structural and functional brain organization into a spatial reference framework. They provide orientation, enable localization, and empower the discovery of underlying structural principles.

Cytoarchitecture describes the characteristics of neuronal cell distributions, which can be analyzed in microscopic scans of cell-body stained histological sections acquired from postmortem human brains. Analysis of cytoarchitecture enables the segregation of the human brain into architecturally distinct areas, which are indicators for connectivity and function (Zilles et al., 2015). As such, creating cytoarchitectonic brain atlases is crucial to identify functional, genetic, physiological, molecular, and connectivity data with microstructurally defined entities (Amunts et al., 2015).

The high inter-individual variability between different brains (Von Economo, 1925) makes it necessary to consider multiple brains for the creation of a cytoarchitectonic brain atlas. The established method for mapping cytoarchitectonic brain areas in histological sections (Schleicher et al., 1999) is precise and reliable, but incurs a significant manual overhead, preventing its application to many sections and brain samples. This limitation motivates the development of methods that provide a higher degree of automation and thereby enable handling the ever-growing amount of image data available from high-throughput microscopy.

Previous work on automated cytoarchitecture analysis (Spitzer et al., 2017, 2018b; Spitzer, 2020) has demonstrated that deep learning methods have the potential to enable automated cytoarchitectonic brain mapping. In recent years, deep learning methods have been successfully applied to achieve promising results in a variety of different tasks, ranging from image classification (Krizhevsky et al., 2012) and segmentation (Ronneberger et al., 2015), over natural language processing (Vaswani

et al., 2017; Devlin et al., 2018; Brown et al., 2020) and protein structure prediction (Jumper et al., 2021) to playing Go (Silver et al., 2016) and controlling autonomous agents in virtual environments (Mnih et al., 2013; Vinyals et al., 2019).

In this thesis, we aim to take the next step towards automated cytoarchitectonic brain mapping at large scale. Building upon the work by Spitzer et al. (2017, 2018b) and Spitzer (2020), we aim to develop deep learning algorithms that are capable of identifying many cytoarchitectures in many sections from multiple human brains.

## 1.1 Problem statement & objectives

This thesis describes novel methods for automated cytoarchitectonic brain mapping based on high-resolution microscopic scans of histological human brain sections at large scale. It builds upon existing work by Spitzer et al. (2017, 2018b) and Spitzer (2020), who demonstrated the feasibility of deep learning for classifying cytoarchitectonic areas from the visual system. We aim to extend this foundational work by focusing on large-scale applications that consider large series of brain sections from multiple brains and more brain areas from different parts of the human brain. We further aim to provide methods to actively support the mapping workflow and enable large-scale analyses that are not possible using established methods (Schleicher et al., 1999). While optimizing the classification performance of developed methods is a major aspect of this work, we also examine how the developed methods make decisions and behave in different practically relevant application scenarios.

Based on the above problem statement, we formulate the following objectives:

- **Implement a practically applicable workflow for interactive mapping of individual brain areas in large series of sections.** Available methods for cytoarchitectonic mapping (Schleicher et al., 1999) are too time and labor intensive to be applied to large series of histological brain sections. To address this shortcoming, we evaluate specifically designed deep learning models that achieve high classification performance across large series of sections with minimal human interaction.

- **Develop deep learning models for large-scale classification of many different brain areas.** Spitzer et al. (2017, 2018b) and Spitzer (2020) have demonstrated the feasibility of deep learning for mapping a small set of cytoarchitectonic areas from the visual system. We use contrastive learning to extend this work to a significantly larger number of brain areas, representing the first step towards automated cytoarchitectonic mapping at the whole-brain level.

- **Enable efficient integration of topological and contextual information into the classification.** Existing approaches for automated cytoarchi-

tectonic mapping base their predictions on local information extracted from high-resolution image patches. We reformulate the task into a graph node classification problem and apply graph neural network methods to integrate comprehensive topological and contextual information into the classification process.

## 1.2 Outline

Following this introduction, Chapter 2 provides foundations on structural brain organization and deep learning. Section 2.1 introduces the basics of the macroscopic and microscopic structure of the human brain (Section 2.1.1), and provides an overview of brain imaging (Section 2.1.2), brain atlases (Section 2.1.3) and cytoarchitecture (Section 2.1.4). Section 2.2 introduces deep learning methods (Sections 2.2.1 to 2.2.6) relevant to this thesis and gives an overview of exiting work on cytoarchitecture analysis with deep learning (Section 2.2.7).

Chapter 3 describes the data used in the scope of this thesis. Section 3.1 summarizes the histological processing protocol that is applied to acquire microscopic images of histological brain sections. An overview of the acquired image data and derived data is given in Sections 3.2 and 3.3.

Chapter 4 details the hardware (Section 4.1) and software (Section 4.2) used to conduct computational experiments. Section 4.3 describes challenges and implemented solutions regarding the storage and efficient access of the large image datasets.

Chapters 5 to 7 introduce and evaluate three methods for automated identification of cytoarchitectonic brain areas in histological human brain sections. The methods address different challenges of automated cytoarchitectonic brain mapping:

- In Chapter 5, we present a method for interactive mapping of individual cytoarchitectonic areas. The method provides a practical workflow for automated brain mapping across many sections. We propose local segmentation models, specialized deep learning models that trade generalizability for accuracy by focusing on a specific area and specific brain region. The performance of local segmentation models is evaluated for different brain areas and different brains. We demonstrate the practical relevance of the method by computing high-resolution reconstructions of cytoarchitectonic areas and verifying their anatomical plausibility. We further present a developed web interface that allows the application of the method without requiring advanced technical knowledge.

- Chapter 6 addresses the task of training a deep learning model for accurate prediction of many cytoarchitectonic areas in multiple brains. We adapt the

idea of contrastive learning, which has been shown to achieve promising results for image analysis tasks (Chen et al., 2020; Khosla et al., 2020), and apply it to extract cytoarchitectonic features from high-resolution image patches. We evaluate the method in different application scenarios, analyze the behavior of trained models, and identify limitations of the approach.

- In Chapter 7, we propose a method that mimics existing brain mapping workflows by integrating 3D brain topology and contextual information into the classification process. Building upon the results from Chapter 6, we model brain mapping as a node classification task in an attributed graph that represents the cortical surface of a reconstructed brain. We address the newly formulated task using graph neural networks. We study and discuss the performance of graph neural networks in combination with different types of contextual features.

Finally, Chapter 8 discusses the methods and results presented in this thesis and identifies future research directions. The glossary lists abbreviations, mathematical notation, and terms used in this thesis.

# 2 Background

## 2.1 Human brain organization

This section gives an overview of the anatomical structure of the human brain, its cytoarchitectonic organization, and analysis techniques, with a focus on aspects that are relevant in the scope of this thesis. Further information on the presented topics can be found in Von Economo (1925), Creutzfeldt (1995), Zilles et al. (2012), Zilles et al. (2013), Zilles et al. (2015), and Trepel (2017).

### 2.1.1 Human brain structure at multiple scales

The human brain can be grossly subdivided into the brainstem, cerebellum, and cerebrum, of which the latter is of particular interest in the scope of this thesis (Figure 2.1). The cerebrum consists of two hemispheres. They are connected by a strong bundle of nerve fibers (the corpus callosum), enabling information exchange between hemispheres. The surface of the cerebrum is heavily folded (gyrification), which significantly increases the surface area and limits the volume occupied in the skull. Fissures (inward folds) are referred to as sulci, while convolutions (outward folds) are referred to as gyri. Using sulci and gyri as anatomical landmarks, the surface of the cerebrum can be subdivided into lobes: The occipital lobe, the parietal lobe, the temporal lobe, and the frontal lobe. While some primary gyri and sulci (e.g., the *parieto-occipital sulcus* separating parietal and occipital lobes, or the *central sulcus* separating parietal and frontal lobes) can be found in every individual, there is a generally high variability in the gyrification across individuals.

The cerebrum can be subdivided into two types of tissue, which are referred to as gray matter and white matter. The gray matter is primarily composed of nerve cells (neurons), which receive or emit chemical and electrical signals. In contrast, the white matter is composed of thin nerve fibers, which enable transmission of signals between neurons in different parts of the gray matter. Together, they form a complex network of interconnected neurons, enabling the brain to function.

The cerebral gray matter can be subdivided into subcortical gray matter and cortical gray matter. Subcortical gray matter is located inside the cerebrum, where it forms structurally distinct clusters (nuclei). Examples for subcortical nuclei are the thalamus and the amygdala.

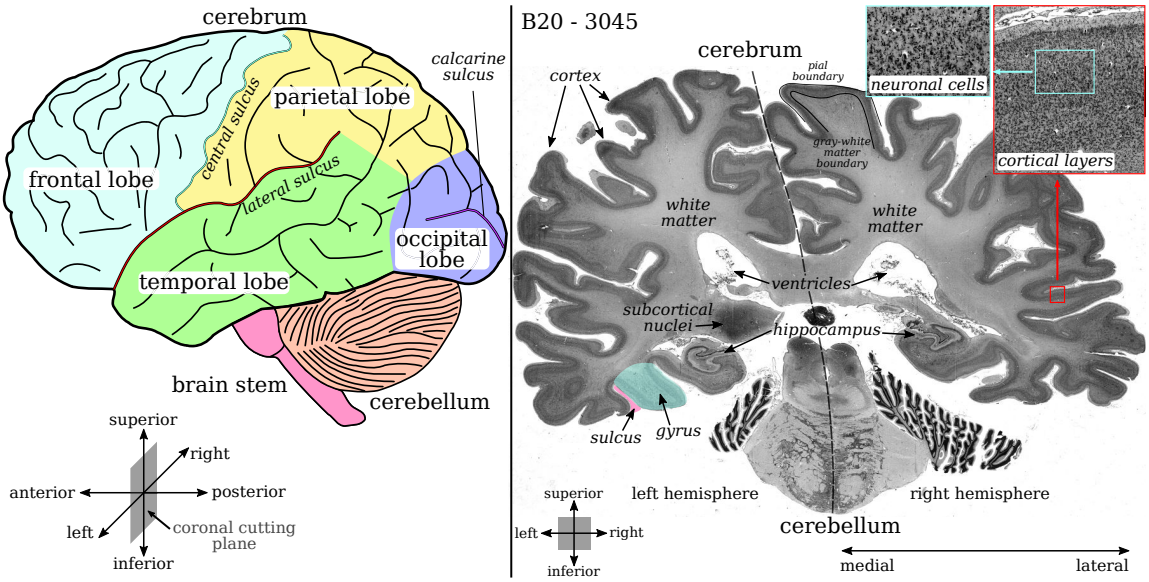



**Figure 2.1:** Structural subdivision of the human brain across multiple scales. **Left**: Schematic illustration of macroscopic brain structure. The cerebrum is characterized by its highly convoluted surface, consisting of sulci and gyri. Some of them (e.g., the *central sulcus*) define anatomical landmarks, which subdivide the cerebrum into lobes. Image modified from https://pixabay.com, under CC0 licence. **Right**: Coronal brain section from B20 (Amunts et al., 2013) with annotations of important anatomical structures. Red and blue boxes show patches extracted from the cortex which illustrate the composition of the cortex across different scales. Annotated orientation axes introduce anatomical directions (left to right, posterior to anterior, inferior to superior, lateral, medial). Image from brain B20 (Chapter 3).

The focus of this work lies on the cortex, a thin ribbon of gray matter that extends across almost the entire surface of the cerebrum. Depending on the location in the brain, the cortical ribbon has a thickness of 2 mm to 4 mm (Von Economo, 1925; Zilles et al., 2012). The outwards facing boundary of the cortex is referred to as pial boundary, while the boundary between cortex and white matter is referred to as gray-white matter boundary. The cellular architecture of the cortex is characterized by cortical layers (lat. *lamina*, *laminae*), which extend approximately parallel to the brain surface, and by the arrangement of cells into vertical *cortical columns*. Approximately 96% (Blinkov et al., 1968; Stephan et al., 1975) of the cortical surface (the isocortex) is composed of six cortical layers, while the remaining regions (the allocortex) show higher variability in layer composition. In the isocortex, cortical layers are numbered with Roman numerals, starting with layer I at the pial boundary and increasing towards layer VI at the gray-white matter boundary.

While the isocortex shows an overall six-layered structure, there are regional differences in the distribution, orientation, size, and type of neurons in the individual

cortical layers. These characteristic properties define the cytoarchitecture of the cortex. Based on cytoarchitecture, the cortex can be subdivided into cytoarchitectonic areas with similar cytoarchitectonic patterns.

### 2.1.2 Neuroimaging

Neuroimaging techniques enable the study of the structural and functional organization of the human brain. Imaging techniques like magnetic resonance imaging (MRI) and computed tomography (CT) are applicable in living organisms (in-vivo) and are nowadays routinely applied in hospitals, where they enable non-invasive imaging of different organs to aid diagnosis. Other in-vivo imaging techniques can capture dynamic processes inside an organism. For example, functional magnetic resonance imaging (fMRI) measures the functional activity in different regions of the brain to give insights into the involvement of different brain regions when performing different tasks.

In-vivo imaging techniques play an essential role in medical diagnosis and research. However, their maximum spatial resolution (i.e., the physical size that each voxel or pixel represents) ranges from several millimeters per voxel down to several hundred micrometers per voxel (Duyn, 2012), which is not sufficient to capture the fine microstructural patterns that are relevant for identifying cytoarchitectonic areas. Research on the cellular organization of the human brain thus relies on ex-vivo imaging approaches, which operate on tissue that has been removed from a living organism.

The microstructural analysis of whole human brains is performed by cutting postmortem human brains acquired through body donor programs into thin histological brain sections (Amunts et al., 2013, 2020). The resulting brain sections can be analyzed using various imaging techniques to highlight specific properties of the tissue. For example, brain tissue can be stained for neuronal cell bodies to highlight the cellular composition, which enables the identification of cytoarchitectonic areas. Alternative techniques include polarized light imaging (PLI) for analyzing nerve fiber distributions, receptor autoradiography for analyzing chemical receptor densities, or immunohistochemistry for the analysis of different cell types. Prepared histological brain sections can be imaged using high-resolution microscopes, which provide image data with significantly higher resolution than in-vivo methods.

The ability to capture different structural properties at high spatial resolution makes brain analysis based on histological brain sections an indispensable tool for brain research. These advantages come at the cost of more complicated, time-consuming, and error-prone image acquisition workflows, less control over the study population, and sometimes incomplete knowledge on non-diagnosed pathologies. In addition, 3D reconstruction methods need to be applied to recover the original 3D structure of the brain from the imaged brain sections. Nevertheless, microscopic

analysis of histological brain sections remains the gold standard for cellular brain analysis, as it is the only method capable of providing the necessary details for microstructural analyses.

### 2.1.3 Human brain atlases

Atlases of the human brain represent important tools for neuroscience research and clinical applications. Similar to geographical atlases, brain atlases embed different types of information (e.g., from different imaging methods) into a reference coordinate system, enabling localization, navigation, and correlation of different data modalities. A brain atlas allows studying relationships between different imaging modalities, e.g., the relationship between brain structure and associated function. In clinical applications, atlases containing information on structural and functional areas in the brain can enable better planning of surgical operations, e.g., to estimate the effect of removing a malign tumor in a specific brain region.

Brain atlases come in different variants, each having its focus, advantages, and shortcomings[1]. One major aspect distinguishing different brain atlases is the data modality they are based on, e.g., MRI (Collins et al., 1994; Evans et al., 2012; Jenkinson et al., 2012; Van Essen et al., 2013), fMRI (Jenkinson et al., 2012), or histology (Schleicher et al., 2005; Zilles et al., 2010; Amunts et al., 2013; Ding et al., 2016; Amunts et al., 2020). The underlying modalities of an atlas determine what kind of analyses can be conducted using the atlas information. The widespread availability and adaptation of MRI scanners makes it possible to base atlases on large and controlled subject populations, allowing them to capture inter-individual variability. However, the limited spatial resolution of MRI limits the use of MRI atlases for fine-grained analysis. In comparison, histology-based cytoarchitectonic atlases (Amunts et al., 2013; Ding et al., 2016; Amunts et al., 2020) provide detailed information on cellular architecture, but require histological processing and can thus only be created based on a much smaller number of subjects. The automation of cytoarchitecture analysis at large scale addressed in this thesis represents an important step towards the creation of cytoarchitectonic atlases based on larger numbers of subjects.

When working with brain atlases, it is important to take the *coordinate space* of atlas data into account. Atlases defined in different spaces (i.e., using different coordinate systems) must be aligned before their relationship can be investigated. In practice, *reference spaces* (also called *template spaces*) enable interoperability between atlases (Holmes et al., 1998; Fonov et al., 2011). Software toolboxes (Eickhoff et al., 2005; Avants et al., 2009; Fischl, 2012; Jenkinson et al., 2012) offer tools to transform data into reference spaces or directly provide atlases in different reference spaces. The following sections list several human brain atlases used in this work.

[1] Amunts et al. (2015) provide a concise overview of standard brain atlases and their properties.

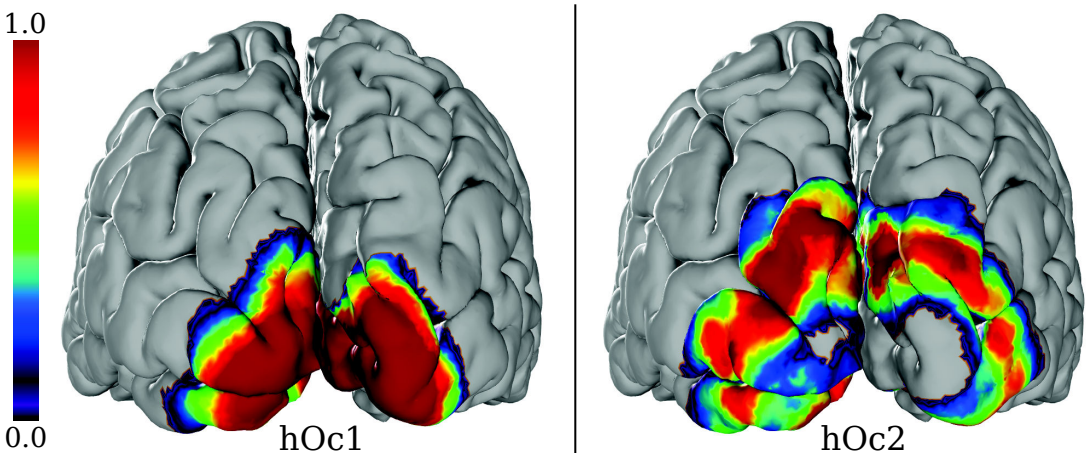



**Figure 2.2:** Probabilistic cytoarchitectonic maps of areas `hOc1` (left) and `hOc2` (right) from the Julich-Brain probabilistic cytoarchitectonic atlas (Amunts et al., 2020). Colors encode the probability of an area being present at the respective location in the brain. Probabilistic maps are computed by superimposing annotations from 23 postmortem brains in a reference space. Images taken from the *Julich-Brain viewer* (Mohlberg et al., 2012).

### The Julich probabilistic brain atlas

Julich-Brain (Amunts et al., 2020) is a probabilistic atlas of the human brain (Figure 2.2). It is based on annotations of cytoarchitectonic areas in histological sections of 23 postmortem brains. The annotations were transformed into a reference space, where they were superimposed to create *probabilistic maps* of cytoarchitectonic areas. Each voxel of a probabilistic map specifies the probability of an area being present at the respective location. The aggregation across multiple subjects bridges the gap between subject-specific and subject-independent maps, which allows making more general and statistically profound statements on cytoarchitecture. As new areas and subjects are integrated, the statistical reliability of the atlas improves. Thus, the development of automated mapping methods contributes to improving probabilistic atlases.

### The BigBrain cytoarchitectonic atlas

BigBrain (Amunts et al., 2013) is an ultrahigh-resolution 3D model of an adult human brain with an isotopic resolution of 20 µm per voxel (Figure 2.3). It was created by reconstructing 7404 microscopic images of cell-body stained histological brain sections into a consistent 3D volume. Its high resolution and anatomical detail enable the integration of multimodal data with the underlying microstructure. The 3D context available for BigBrain further allows the analysis of structures that cannot be studied to a satisfactory degree in two-dimensional microscopic images.

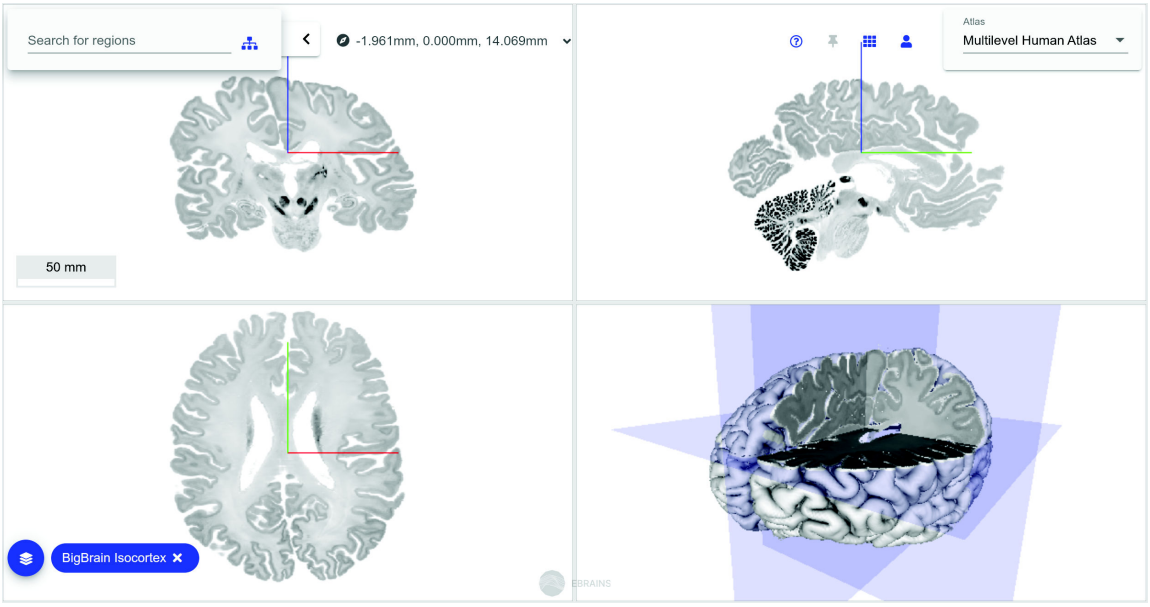



**Figure 2.3:** BigBrain (Amunts et al., 2013) high-resolution cytoarchitectonic human brain atlas, created by 3D reconstructing 7404 microscopic images of cell-body stained histological brain sections into a consistent 3D volume (20 µm per voxel). The top-left image pane shows the original coronal image plane. Top-right and bottom-left image panes show sagittal and horizontal virtual cross-sections through the reconstructed brain volume, respectively. The bottom-right pane shows a 3D view of the reconstructed volume. Image taken from the *EBRAINS Interactive Atlas Viewer* (`https://atlases.ebrains.eu/viewer`).

#### The Allen Adult Human Brain Atlas

The Allen Adult Human Brain Atlas (AAHA) (Ding et al., 2016) is a multimodal human brain atlas. It comprises MRI and diffusion weighted MRI (DWI) measurements of an adult human brain before sectioning, as well as microscopic scans of 1356 histological brain sections with a thickness of 50 µm. 679 of these sections were stained for cell bodies using Nissl staining and subsequently digitized at 1 µm/px resolution. Anatomical structures (including cytoarchitectonic areas) were then digitally annotated in a subset of 106 unevenly spaced sections.

### 2.1.4 Cytoarchitecture

Cytoarchitecture describes the spatial organization of neuronal cells, including the distribution, orientation, and type of the cells. It provides an important microstructural reference for connectivity and function and represents a key aspect for the creation of human brain atlases (Eickhoff et al., 2007; Amunts et al., 2015).

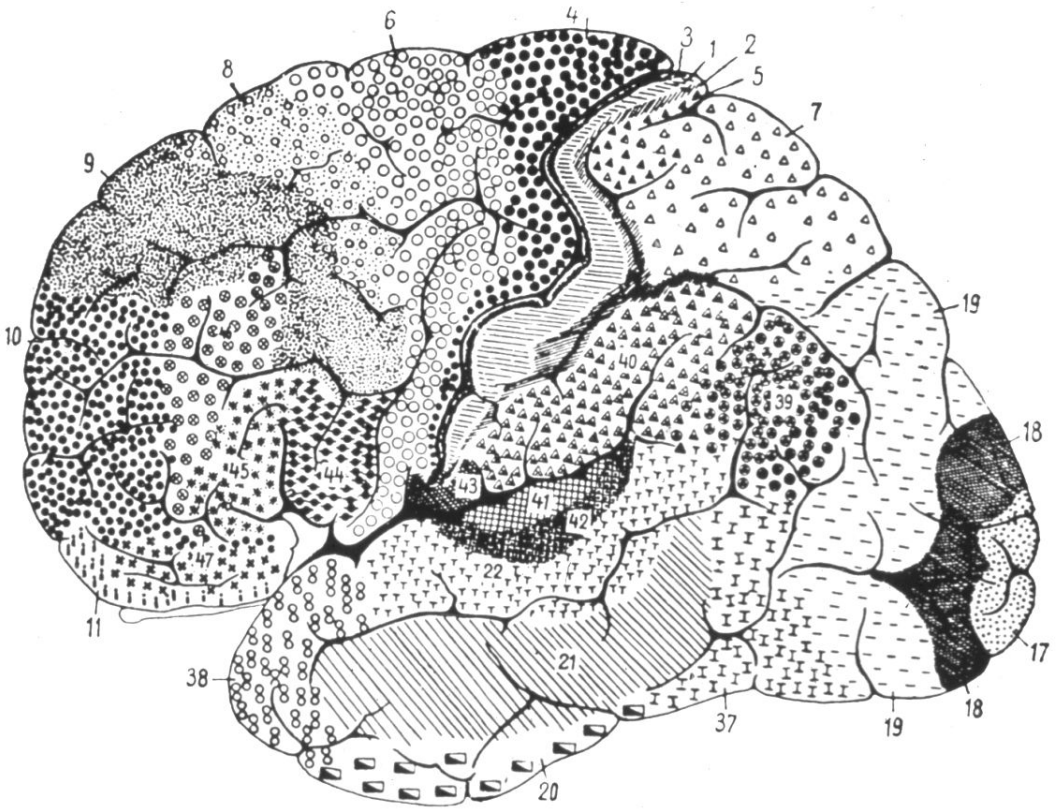



**Figure 2.4:** Lateral view on one of the first cytoarchitectonic maps of the human brain, published by Korbinian Brodmann in 1909. Microstructurally distinct cytoarchitectonic areas are marked by different patterns and numbered. The numbering scheme introduced by Brodmann is partly still used today (Table 3.2). Image from Brodmann (1909, p. 131).

### Cytoarchitectonic areas

As mentioned in Section 2.1.1, the isocortex is almost everywhere comprised of six cortical layers. Cortical layers are identified by Roman numerals I to VI, starting with layer I at the pial boundary. The layers of the isocortex differ in their cellular composition (Figure 2.5). Although the composition varies strongly in different regions of the brain (which is the basis for the distinction of cytoarchitectonic areas) and across brains, several characteristic organizational principles can be observed for each layer (Von Economo, 1925; Zilles et al., 2012):

**Layer I (molecular layer)** This outer cortical layer contains only few scattered neurons and consists primarily of dendrites originating from neurons in deeper layers.

**Layer II (outer granular layer)** This layer is comprised of small and densely packed pyramidal cells. Pyramidal cells are named based on their characteristic triangular shape.

**Layer III (outer pyramidal layer)** This layer consists of small to medium-sized pyramidal cells, which increase in size towards deeper layers and are less densely packed compared to layer II.

**Layer IV (inner granular layer)** This layer contains densely packed cells and is comprised of granular cells, small pyramidal cells, and fusiform cells. The latter cell type summarizes different cell types with varying (polymorph) appearances. Layer IV can often be further subdivided into multiple sublayers, especially in sensory areas. Sublayers are denoted by a letter suffix, e.g., layer IVa, IVb, IVc.

**Layer V (inner pyramidal layer)** This layer is characterized by large pyramidal cells and a medium cell packing density.

**Layer VI (fusiform layer)** The deepest layer contains cells of varying appearance (fusiform cells), as well as few larger pyramidal cells. The cell density decreases as the cortex transitions into white matter.

The analysis of regional differences in the specific cellular and laminar composition in the cortex allows its subdivision into cytoarchitectonic areas. This process of subdividing the cortex into distinct areas is referred to as *cytoarchitectonic brain mapping*[2]. A pioneer of cytoarchitectonic analysis was Korbinian Brodmann, who formulated criteria for the classification of cytoarchitectonic areas and proposed one of the most influential cytoarchitectonic maps of the human brain (Brodmann, 1909, Figure 2.4). His work provides the basis for many subsequent works (Vogt et al., 1919; Von Economo, 1925; Amunts et al., 2015). The analysis of cytoarchitectonic areas has become an active field of research, intending to improve our understanding of the brain's microstructural organization (Amunts et al., 2015; Zilles et al., 2015; Amunts et al., 2020).

### Structure and function

Cytoarchitectonic areas are correlated with specific brain functions (Eickhoff et al., 2007; Zilles et al., 2012): Primary sensory areas directly receive sensory signals from the corresponding sensory organs, e.g., the primary visual and auditory areas for processing of external visual and auditory stimuli, respectively. In contrast, higher

[2] In this work, the term *brain mapping* refers to *cytoarchitectonic brain mapping*, if not mentioned otherwise.

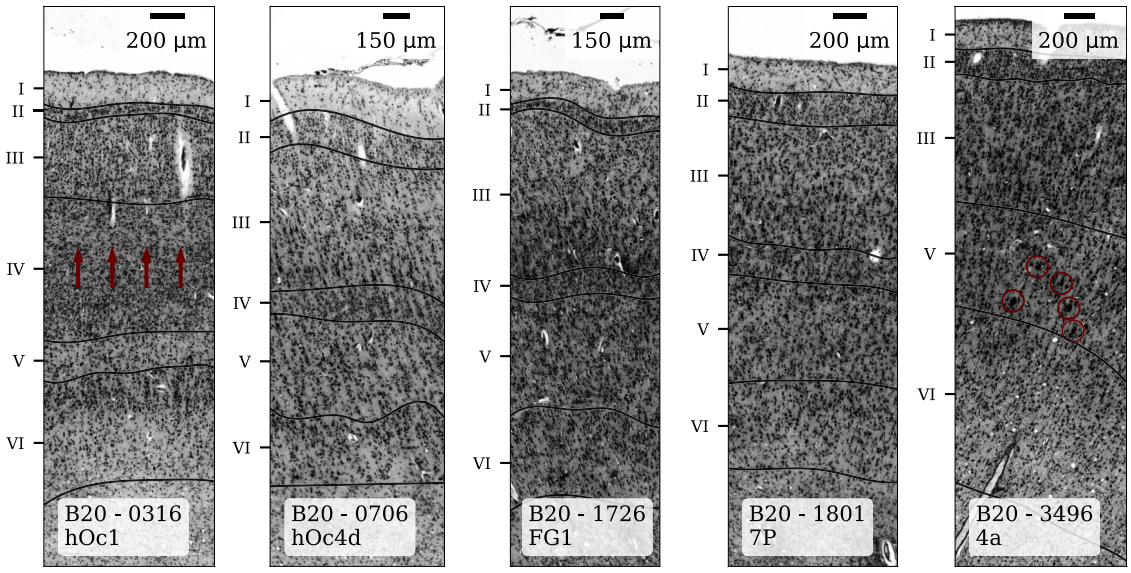



**Figure 2.5:** Image patches extracted from different cytoarchitectonic areas. Black lines delineate the borders between cortical layers. Cytoarchitectonic areas are characterized by their cellular composition, e.g., the size, shape, and distribution of cells, and their arrangement into cortical layers (Amunts et al., 2020). Red arrows mark the characteristic stripe of Gennari within layer IV of visual area `hOc1`, which results from a high density of nerve fibers entering the visual system. Red circles indicate the position of some Betz giant cells in the primary motor area `4a`, which project into the spinal cord to control motor functions. Image patches and cortical layer annotations for `hOc1`, `hOc4d`, `FG1`, `7P` and `4a` are based on EBRAINS datasets (Dickscheid et al., 2021a,b,c,d,e).

unimodal sensory areas interpret signals arriving from the corresponding lower-level areas. Multimodal association areas connect to other multimodal association areas to combine various levels of sensory information. In addition, some multimodal association areas are connected to motor areas, allowing the initiation of movement in reaction to inner or outer stimuli (Zilles et al., 2012).

While an extensive description of all characteristic properties found in cytoarchitectonic areas is beyond the scope of this thesis, the following examples introduce some principles that illustrate how cytoarchitecture expresses in different areas:

The primary visual cortex `hOc1` (Figure 2.5, left) is an area located in the posterior occipital lobe (Figure 2.1). It is the first cortical area to process visual stimuli received from the eyes. The primary visual cortex is characterized by the stripe of Gennari, a visible bright ribbon that forms a sublayer in cortical layer IV (see red arrows in Figure 2.5). This stripe is formed by a high density of nerve fibers entering this layer to transmit visual stimuli and by densely packed granular cells. In general, primary

sensory areas show a strongly expressed layer IV, where most external stimuli enter the cortex (Zilles et al., 2012).

The primary motor cortex (Figure 2.5, right) receives signals from multimodal association areas to initiate movement. Here, layer IV does not exist in the adult brain (Zilles et al., 2012), as the motor cortex does not receive immediate external stimuli. Instead, layer V, representing the main system for outgoing signals (Zilles et al., 2012), is strongly expressed in the motor cortex. It contains the *Betz giant cells* (see red circles in Figure 2.5), large pyramidal cells with a diameter of up to 100 µm that project into the spinal cord.

While most cytoarchitectonic areas do not express as clear and interpretable cytoarchitectonic features as the primary visual and the primary motor cortex, these examples illustrate the relationship between the cellular organization of cytoarchitectonic areas, their function, and their connectivity. The study of these relationships motivates the systematic and large-scale analysis of cytoarchitectonic areas in the human brain.

### Challenges of cytoarchitectonic brain mapping

Cytoarchitectonic areas are defined by complex microstructural patterns, which can typically only be identified and correctly classified after intensive training. Their complexity makes mapping of cytoarchitectonic areas a challenging and thereby time and labor-intensive task. In addition, the variability between subjects (i.e., brains) introduces another dimension of complexity. The appearance of individual brain areas can vary greatly between individuals (Amunts et al., 2000). The variance can be attributed to slightly different histological processing procedures and the inherent inter-subject variability that makes every human's brain unique. Researchers studying cytoarchitectonic areas need to adapt to this variability, and it has to be expected that the same is true for algorithmic approaches.

Another challenge results from the sectioning process, which projects the three-dimensional structure of the brain onto a series of independent two-dimensional planes (i.e., images). These isolated images cannot capture the full three-dimensional structure of the original brain volume. This effect becomes clear when considering the phenomenon of *obliquely cut regions* (short *oblique cuts*). Oblique cuts are regions of the cortex where the angle between the plane by which the brain was cut (the *cutting plane*, which is equivalent to the image plane) and the local normal vector of the brain surface are not aligned. This issue is schematically illustrated in Figure 2.6 (left): When the normal vector of the brain surface and the cutting plane align (green line), all cortical layers and their proportions are properly captured in the projection. However, as the angle between the cutting plane and the normal vector increases, some layers are not covered in the projection, and their relative proportions are skewed

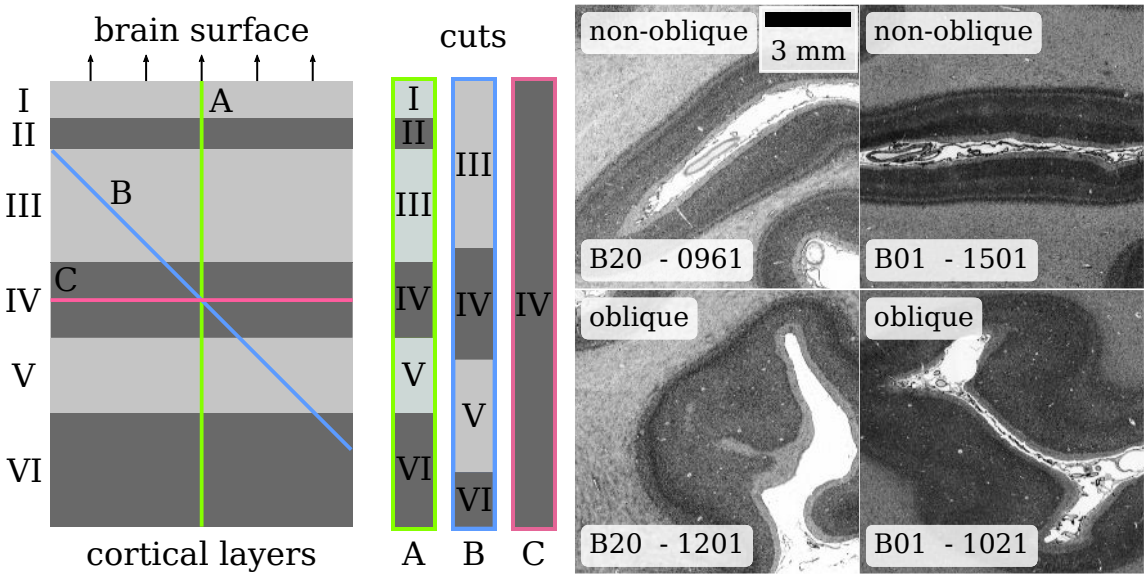



**Figure 2.6:** Illustration of the oblique cut issue occurring in serially cut brain sections. **Left**: Schematic illustration of the isocortex with its six cortical layers. Depending on the angle of the cutting plane (A, B, C) relative to the normal vector of the brain surface (arrows), the proportion of individual layers is captured well (A), or is increasingly distorted (B, C). If the cutting angle is too oblique, the laminar composition of the isocortex cannot be classified correctly. **Right**: Real-world examples of non-oblique (top row) and oblique (bottom row) cuts from two brains (`B01` and `B20`). Non-obliquely cut tissue appears as a well-shaped band with approximately parallel pial boundary and gray-white matter boundary, while obliquely cut tissue often appears as degenerated "bulges" with variable shape.

(blue and red lines). Figure 2.6 (right) shows real-world examples of this problem: Non-obliquely cut regions of the cortex (top row) appear as a well-shaped band with approximately parallel pial boundary and gray-white matter boundary. In contrast, obliquely cut regions (bottom row) are often expressed as degenerated "bulges" with variable shapes.

The problem of obliquely cut tissue was already described early by Von Economo (1925), who identified them as an inherent shortcoming of the serial brain sectioning method that could only be overcome by using more sophisticated and complex sectioning techniques. In practice, such ambiguities are resolved by taking context information into account, e.g., by inspecting adjacent sections or regions within the same sections with less oblique cutting angles.

### Observer-independent brain mapping

Analyzing cytoarchitectonic areas using non-systematic microscopic analysis methods is susceptible to errors arising from the subjective perception of the person performing

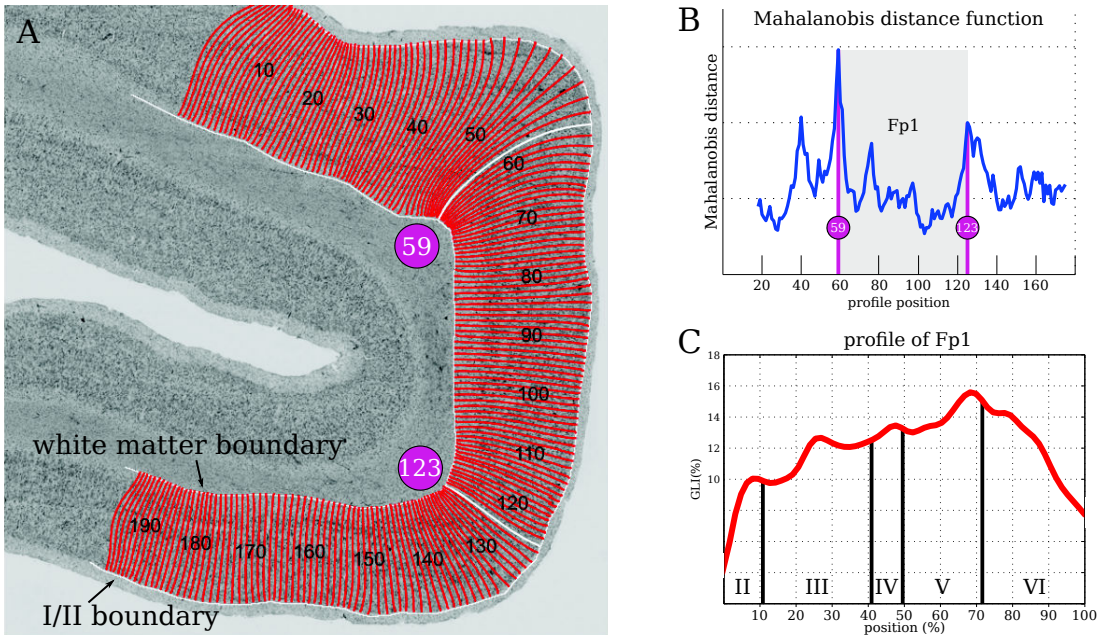



**Figure 2.7:** Observer-independent method for cytoarchitectonic boundary detection as proposed by Schleicher et al. (1999). **A**: Cortical traverses along the cortical ribbon are computed based on delineations of the layer I/II boundary and the gray-white matter boundary in a selected region of interest. Sampling the GLI along the traverses results in cortical profiles, which characterize the cellular composition in relation to the depth at each position along the cortex. **B**: Statistically significant changes of the cellular composition are detected by blockwise comparison of adjacent profiles. Profiles are compared based on the Mahalanobis distance (Mahalanobis, 1936). Peaks in the Mahalanobis distance indicate significant changes of cytoarchitecture and thereby possible boundaries between cytoarchitectonic areas. **C**: Average GLI profile of area `Fp1`, which is enclosed by traverses 59 and 123. The average profile can be interpreted as a "fingerprint" of the area. Figure reproduced from Bludau (2011) by permission of the author.

the analysis (*personal bias*). To address this issue, Schleicher et al. (1999) propose an observer-independent method for identifying possible cytoarchitectonic boundaries in a reproducible manner (Figure 2.7). This method is considered the current gold standard for delineation of cytoarchitectonic areas and represents a reference for automated brain mapping methods.

The method proposed by Schleicher et al. (1999) aims to detect significant changes in the cellular composition of the cortex, as these are indicators for areal borders. In a first step, the method extracts cortical traverses between the layer I/II boundary (i.e., the boundary between cortical layers I and II) and the gray-white matter boundary. Delineations of the layer I/II boundary and the gray-white matter boundary need to be provided manually. The cortical traverses are constructed by integrating

equidistant points in the cortex through a Laplacian potential field (Jones et al., 2000) between the two delineations (Figure 2.7, A).

The constructed cortical traverses are used to sample from the gray level index (GLI) (Schleicher et al., 1999). The GLI encodes the local relative fraction of neuronal cell bodies in the cortex. It is computed by segmenting cell bodies in microscopic images at 1 µm/px resolution and computing the volume fraction of cell bodies in 16 px × 16 px windows. The resulting GLI is a gray-level image with 16 µm/px resolution. The sampling of the GLI with the cortical traverses results in one cortical profile per cortical traverse. A cortical profile characterizes the cellular composition at a specific position in the cortex (Figure 2.7, C).

The created cortical profiles are used to detect statistically significant changes in the cellular composition, which indicate possible borders between cytoarchitectonic areas (Figure 2.7, B). To achieve this, each profile is condensed into a 10-dimensional feature vector encoding its central moments and their derivatives. The resulting feature vectors are compared using the Mahalanobis distance (Mahalanobis, 1936). The comparison is performed blockwise to increase the robustness of the detection step. Statistically significant changes are detected using a t-test with Bonferroni correction.

The presented method has been successfully applied in a range of different studies (Amunts et al., 2020). However, it relies on a manual pre-localization of areal boundaries, as well as precise manual delineations of the layer I/II boundary and the gray-white matter boundary to compute the cortical traverses. These requirements limit the applicability for brain mapping at a large scale (i.e., when considering large numbers of cytoarchitectonic areas and brain sections) and motivate the development of automated methods pursued in this thesis.

## 2.2 Deep learning methods

Deep learning is a machine learning method that has been shown to perform well in tasks involving the analysis of high-dimensional and complex data. Its good performance for image analysis tasks (Krizhevsky et al., 2012; Ronneberger et al., 2015; Redmon et al., 2016) makes deep learning a promising technique for automated cytoarchitecture classification (Section 2.2.7).

This section introduces the deep learning methods that are used in the scope of this thesis. It gives an overview of relevant concepts and methods, with a particular focus on their role for the classification of cytoarchitectonic areas. A comprehensive discussion of fundamental and advanced deep learning methods can be found in Goodfellow et al. (2016).

Deep learning has a long history[3], which dates back much longer than the rather recent works (Krizhevsky et al., 2012) that led to the large interest and widespread adoption of deep learning. The "perceptron" proposed by Rosenblatt (1958) can be seen as one of the earliest predecessors of modern deep learning algorithms. It was the first model to adjust its parameters by looking at example inputs and thereby "learn" how to solve given problems. Building upon this pioneering work, Fukushima et al. (1982) proposed the "neocognitron", a hierarchical, multi-layer neural network that forms the foundation for modern convolutional neural networks (CNNs), which are successfully used for image analysis tasks (Section 2.2.3). Another important milestone in the history of deep learning is marked by the proposal of the backpropagation algorithm by Rumelhart et al. (1986), which enables training of deep neural networks with multiple layers. After two periods of active research in the 1940s-1960s (McCulloch et al., 1943; Hebb, 1949; Rosenblatt, 1958) and 1980s-1990s (Rumelhart et al., 1986), deep learning and its predecessors were largely ignored. This phase of widespread inactivity in the field (also referred to as "AI winter") ended with the introduction of efficient and thereby practically applicable training techniques (Hinton et al., 2006; Krizhevsky et al., 2012), as well as the increasing availability of large datasets (Russakovsky et al., 2015) and computational resources (Krizhevsky et al., 2012; Abadi et al., 2016; Al-Rfou et al., 2016; Paszke et al., 2019). In recent years, the ongoing renaissance of deep learning led to breakthroughs in application areas such as image classification (Krizhevsky et al., 2012; Dosovitskiy et al., 2020), image segmentation (Long et al., 2015; Ronneberger et al., 2015), object detection (Redmon et al., 2016), autonomous driving (Grigorescu et al., 2020), natural language processing (Vaswani et al., 2017; Devlin et al., 2018; Brown et al., 2020), image generation (Goodfellow et al., 2014; Ramesh et al., 2021), protein structure prediction (Senior et al., 2020; Jumper et al., 2021), or control of autonomous agents in virtual environments (Mnih et al., 2013; Silver et al., 2016; Vinyals et al., 2019). In 2018, Yoshua Bengio, Geoffrey Hinton, and Yann LeCun were awarded the ACM A.M. Turing Award for their major contributions to the field of deep learning, demonstrating the important role of deep learning for the field of computer science.

### 2.2.1 Supervised training of deep neural networks

*Supervised machine learning* algorithms aim to recover the relationship between an input and an output variable based on a set of examples consisting of inputs and corresponding outputs. Typical supervised learning tasks are *classification*, where

[3]The presented summary provides a brief, but in no way complete overview of selected milestones in the history of deep learning. Cited references provide examples of influential work in the field, but should not be seen as a complete list of all relevant publications. See Goodfellow et al. (2016) for a more detailed discussion of deep learning history.

the goal is to assign a given data sample (e.g., an image) to a specific *class* (e.g., the type of object visible in an image), or *regression*, where the goal is to predict a continuous value for each data sample (e.g., price, temperature). Cytoarchitectonic mapping can be modeled as a classification task (Spitzer et al., 2017, 2018b): Given an image, a supervised machine learning algorithm is tasked to predict the corresponding cytoarchitectonic area. Their ability to learn complex relationships from examples makes supervised learning methods well suited for cytoarchitecture classification. This section focuses on the supervised learning setting. A discussion of other learning paradigms is given in Section 2.2.4.

Supervised learning can be formulated as an optimization problem

$$\arg\min_{\theta \in \Theta} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{X}} \left[ l(f(\boldsymbol{x};\theta), \boldsymbol{y}) \right] \tag{2.1}$$

where $f$ is a function (also referred to as *model*) parameterized by $\theta \in \Theta$ that maps an input[4] $\boldsymbol{x}$ to a prediction $\hat{\boldsymbol{y}} = f(\boldsymbol{x};\theta)$, $l$ is a loss function measuring the error between a prediction $\hat{\boldsymbol{y}}$ and the target value $\boldsymbol{y}$ (also known as *groundtruth* or *labels*), and $\mathcal{X}$ is the joint data distribution of input samples $\boldsymbol{x}$ and corresponding target values $\boldsymbol{y}$. The expected loss in Equation 2.1 is estimated by the mean over a *training dataset* $\mathbb{X} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) \mid 1 \leq i \leq n\}$ containing $n$ samples from $\mathcal{X}$:

$$\hat{\theta} = \arg\min_{\theta \in \Theta} L(\mathbb{X};\theta) = \arg\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} l(f(\boldsymbol{x}_i;\theta), \boldsymbol{y}_i) \tag{2.2}$$

In machine learning, the optimization phase is referred to as training or *learning* phase, and the model is said to *learn* how to solve the given task.

The term "deep learning" summarizes a group of machine learning algorithms that are based on deep neural networks. Formally, a deep neural network is a differentiable non-linear functional mapping that is parameterized by a set of parameters $\theta$. Deep neural networks are often conceptualized and communicated as compositions of so-called layers (Section 2.2.2). A layer is a smaller functional module that applies a parameter-dependent transformation to the incoming data before passing it on to the next layer. Different kinds of layers are used for different tasks, application scenarios, and different kinds of data (e.g., images, graphs). The layer composition of a deep neural network is referred to as its architecture (Section 2.2.3), and the number of layers is referred to as depth. A goal of deep learning research is to find suitable architectures and training strategies for different tasks. The performance of different architectures and training strategies is assessed using *evaluation met-*

[4]For sake of brevity, this section assumes vector-valued input and output variables. However, all definitions apply to higher-dimensional data (e.g., images) as well.

*rics* (Section 2.2.1), which measure how well a deep neural network performs for a given task.

This section introduces optimization procedures, loss functions, and evaluation methods for supervised deep learning, with a focus on methods used in this work. Sections 2.2.2 and 2.2.3 describe neural network layers and architectures used in this thesis. Sections 2.2.4 and 2.2.5 introduce unsupervised learning and visual representation learning methods relevant to this work. Section 2.2.6 details technical aspects of deep neural networks training. Section 2.2.7 discusses existing applications of deep learning for cytoarchitecture analysis.

### Optimization

The optimization problem in Equation 2.2 is typically solved using gradient descent or one of its variants (Sutskever et al., 2013; Kingma et al., 2014). Although gradient descent is not guaranteed to find global minima for non-convex optimization problems[5], it has been successfully applied in many deep learning applications and has become the de-facto standard for training deep neural networks. Intuitively, gradient descent is often described using the metaphor of a ball rolling down a hill and into a valley, always following the direction of the steepest slope (the negative gradient direction). Here, the ball stands for a position in the parameter space, the hill represents the image space of the loss function (also referred to as *loss landscape*), and the valley represents a (local) minimum of the loss function. In its basic formulation, gradient descent iteratively updates the parameters $\theta$ by making small adjustments in the direction of the negative gradients of the loss function $L$ with respect to the parameters $\theta$

$$\theta^{(i+1)} = \theta^{(i)} + \Delta\theta^{(i+1)}, \tag{2.3}$$

$$\text{with } \Delta\theta^{(i+1)} = -\lambda \nabla_\theta L\left(\mathbb{X}; \theta^{(i)}\right) \tag{2.4}$$

$$= -\lambda \frac{1}{n} \sum_{j=1}^{n} \nabla_\theta \, l\left(f\left(\boldsymbol{x}_j; \theta^{(i)}\right), \boldsymbol{y}_j\right), \tag{2.5}$$

where $\theta^{(i)}$ specifies the parameter configuration at iteration $i$, starting from a random (He et al., 2016a) initial parameter configuration $\theta^{(0)}$. $\lambda \in \mathbb{R}^{\geq 0}$ is referred to as the *learning rate* and specifies the step size taken in each iteration. The learning rate is often adjusted over the course of the training using different strategies. For example, the learning rate is often decreased as the training progresses. The intuition

[5] In most cases, training a deep neural network poses a non-convex optimization problem. One reason for this is that deep neural network are often *non-identifiable*, i.e., there are different parameter configurations that result in equivalent models and consequently multiple local minima (Goodfellow et al., 2016).

behind this strategy is to take large steps early on to explore the loss landscape, while taking smaller steps later during training to descent into a good (local) minimum.

The learning rate is an example for a *hyperparameter*. The term hyperparameter summarizes all parameters that control the behavior of the deep neural network model or the training procedure, but are not included in $\theta$ and are thus not updated during training. The type and number of layers defining the architecture of a deep neural network are other examples for hyperparameters.

**Stochastic gradient descent** In practice, computing the gradients in Equation 2.3 across the entire dataset $\mathbb{X}$ is often computationally expensive and not feasible for large datasets. Stochastic gradient descent (SGD)[6] (Robbins et al., 1951; Kiefer et al., 1952) addresses this issue by estimating gradients based on *batches* $\mathbb{B} \subset \mathbb{X}$ with *batch size* $b = |\mathbb{B}|$. The batch size $b$ is typically chosen to be relatively small compared to the total dataset size $n$. It can be held fixed as the dataset size grows, enabling the processing of datasets containing many samples.

When training with SGD, the dataset $\mathbb{X}$ is subdivided into $\lceil n/b \rceil$ disjoint batches, which are processed sequentially to update the model parameters (Equation 2.3). One pass through the dataset is referred to as an *epoch*, and training encompasses multiple epochs. The composition of batches is randomized between epochs (e.g., by shuffling the training dataset after each epoch). The size of the batches is a hyperparameter.

In addition to its computational benefits, the stochastic nature of SGD can positively affect the convergence behavior (Goodfellow et al., 2016). In practice, estimating the gradient-based on a batch of data often approximates the global gradient (i.e., the gradient with respect to to the entire training dataset) sufficiently well. Thus, SGD can perform more gradient updates than gradient descent in a given time frame and achieve faster convergence.

**Backpropagation** The ability to efficiently compute the gradients of a loss function with respect to the parameters of a deep neural network is crucial for the application of gradient-based optimization methods. The *backpropagation algorithm* proposed by Rumelhart et al. (1986) allows the computation of gradients in a deep neural network by recursive application of the chain rule. The algorithm comprises two steps: In the *forward pass*, input examples are passed forward through the layers of the deep neural network, while storing all intermediate layer outputs for subsequent gradient computation. In the following *backward pass*, gradients are recursively computed, starting from the last layer and moving backwards towards the first layer.

[6] SGD is sometimes also referred to as *minibatch SGD*, which emphasizes that gradient estimation is based on (mini)batches of data.

Computing the gradient for a deep neural network involves many mathematical operations. Thus, the recursive application of the chain rule performed by the backpropagation algorithm leads to many multiplications. The *vanishing gradient problem* arises when these multiplications include small values. The repeated multiplication of small values results in small gradient updates (Equation 2.5) and thus to extremely slow convergence. The related *exploding gradient problem* arises when the multiplications include too large values, as this leads to divergence of the optimization[7]. The use of adequate activation functions (Section 2.2.2) or normalization layers (Section 2.2.2) is crucial to mitigate these problems. A comprehensive discussion of the backpropagation algorithm can be found in (Goodfellow et al., 2016).

**SGD with Nesterov momentum** Training with SGD can lead to large variations in the gradients that are used to update the model parameters. In particular, training with small batch sizes (which is often required due to memory limitations) can lead to noisy estimates of the gradients and consequently poor optimization performance. *Momentum* (Polyak, 1964) is a technique that aims to prevent sudden changes of the gradient direction between iterations, reducing the negative effect of noisy gradients and accelerating training. Speaking in the metaphor of a ball rolling down a hill, momentum adds a certain amount of inertia to the ball, preventing it from uncontrolled movements into various directions and from getting stuck in small local valleys. Using a momentum factor $\mu \in [0, 1]$, the update vector in Equation 2.5 becomes

$$\Delta\theta^{(i+1)} = -\lambda \nabla_\theta L\left(\mathbb{B}; \theta^{(i)}\right) + \mu \Delta\theta^{(i)}. \tag{2.6}$$

Thus, each update is influenced by the gradients, as well as the update direction $\Delta\theta^{(i)}$ of the previous iteration.

*Nesterov momentum* (Sutskever et al., 2013) is a variant of momentum inspired by Nesterov's accelerated gradient method (Nesterov, 1983), which changes the update vector to

$$\Delta\theta^{(i+1)} = -\lambda \nabla_\theta L\left(\mathbb{B}; \theta^{(i)} + \mu \Delta\theta^{(i)}\right) + \mu \Delta\theta^{(i)}. \tag{2.7}$$

Compared to standard momentum (Equation 2.6), gradient descent with Nesterov momentum takes a step $\mu\Delta\theta^{(i)}$ before evaluating the gradient of the loss function. This can be interpreted as taking a SGD step and then correcting the update based on the gradients at the new position. Nesterov momentum has been shown to stabilize training and accelerate convergence (Sutskever et al., 2013).

[7] In practice, vanishing and exploding gradients often result in underflows and overflows of floating-point data types, respectively.

**Regularization** *Overfitting* is a phenomenon observed when a model (e.g., a deep neural network) fits the training data well, but is unable to generalize to new data. It becomes a particular challenge when the available training dataset is small, as a model may fail to capture underlying relationships in the data when the sample size is too small. *Regularization* techniques can counter overfitting and improve the generalizability of models to unseen data. *L2 regularization* (also referred to as weight decay) is often used to train deep neural networks (Goodfellow et al., 2016). It works by adding a regularization term to the loss function to prevent the magnitude of the parameters $\theta_j \in \theta$ from growing indefinitely and fitting the training data perfectly:

$$L_{wd}(\mathbb{B};\theta) = L(\mathbb{B};\theta) + \omega \sum_{\theta_j \in \theta} \left\|\theta_j\right\|_2^2 . \tag{2.8}$$

The weight decay factor $\omega \in \mathbb{R}^{\geq 0}$ controls the impact of the regularization term.

**Layer-wise adaptive rate scaling (LARS)** SGD enables the training with batches that are significantly smaller than the entire training dataset. Hardware for training deep neural networks (Section 2.2.6) enables the parallel processing of training samples in a batch. Thus, using large batch sizes (e.g., as large as the available memory permits) enables the efficient use of available hardware resources. As a larger batch size leads to fewer training updates per epoch, Goyal et al. (2017) propose to increase the learning rate $\lambda$ linearly with the batch size. However, training with large learning rates can lead to unstable training behaviour (Goyal et al., 2017; You et al., 2017).

Layer-wise adaptive rate scaling (LARS) addresses this problem by computing a separate local learning rate $\lambda_j$ for each parameter $\theta_j \in \theta$:

$$\theta_j^{(i+1)} = \theta_j^{(i)} + \Delta\theta_j^{(i+1)} \tag{2.9}$$

$$\text{with } \Delta\theta_j^{(i+1)} = -\lambda_j^{(i)} \nabla_{\theta_j} L\left(\mathbb{B};\theta^{(i)}\right), \tag{2.10}$$

$$\lambda_j^{(i)} = \lambda\, \eta\, \frac{\left\|\theta_j^{(i)}\right\|_2}{\left\|\nabla_{\theta_j} L\left(\mathbb{B};\theta^{(i)}\right)\right\|_2}. \tag{2.11}$$

The *trust factor* $\eta \in [0,1]$ adds additional control over how much the parameters can change per iteration. You et al. (2017) report that the ratio between parameter and gradient magnitudes can vary strongly for different parameters and different layers, leading to instabilities if the learning rate is too high. The local learning rate rescaling mitigates this effect and stabilizes training with large batch sizes. It can be combined with other optimization techniques, e.g., L2 regularization or momentum.

**Data augmentation** Supervised machine learning methods depend on the availability of large labeled training datasets (e.g., data samples and associated labels). Labeling datasets is time and labor-intensive and often requires domain knowledge. *Data augmentation* softens the requirement for large labeled datasets: It randomly transforms the available data and thus artificially creates additional training samples. This approach is particularly useful for image data, as images can be transformed in various ways that preserve their semantic meaning (i.e., the class a sample belongs to). Typical examples for image data augmentation involve rotation, mirroring, or pixel intensity modification. The transformation parameters (e.g., rotation angle) are randomly and independently determined for each data sample during training.

When designing a data augmentation pipeline, it is important to consider the data and the given task. Data augmentations aim to mimic variations that can be realistically expected in the natural data distribution $\mathcal{X}$. It is necessary to ensure that the relationship between the input and the corresponding label is preserved after the transformation. For example, a model aiming to classify images of digits (like in the popular MNIST dataset, LeCun et al., 1998) should not be trained with too strong random rotation, as it can potentially alter the semantic meaning of the image (e.g., change the digit "6" to "9" and vice-versa).

### Loss functions

A loss function measures how well a given deep neural network performs for a given task. Minimizing[8] the loss function using a suitable optimization algorithm (Section 2.2.1) thus reduces the prediction error of a model. The choice of the specific loss function depends on the given task.

The categorical cross-entropy loss is a loss function for multi-class classification tasks (e.g., classification or segmentation of images). Here, the task is to assign each data point to one of $c$ predefined classes. In this setting, the groundtruth $\boldsymbol{y} \in \mathbb{R}^c$ and the model prediction $\hat{\boldsymbol{y}} \in \mathbb{R}^c$ are modelled as $c$-dimensional vectors representing discrete pseudo-probability distributions over $c$ classes, with $\boldsymbol{y}(i)$ and $\hat{\boldsymbol{y}}(i)$ depicting the true and predicted probability of a sample to belong to the $i$-th class ($1 \leq i \leq c$), respectively. $\hat{\boldsymbol{y}}$ is typically normalized using the softmax function

$$\text{softmax}_i(\hat{\boldsymbol{y}}) = \frac{\exp\left(\hat{\boldsymbol{y}}(i)\right)}{\sum_{j=1}^{c} \exp\left(\hat{\boldsymbol{y}}(j)\right)}. \tag{2.12}$$

[8] Since a *loss function* measures the *disagreement* between predicted and target output, it is minimized during training. In contrast, a *reward function* would measure the *agreement* between the two and thus would need to be maximized.

The result can be interpreted as a probability distribution over $c$ classes. The groundtruth $\boldsymbol{y}$ is represented as a *one-hot encoded vector* with

$$\boldsymbol{y}(i) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}, \tag{2.13}$$

where $j$ denotes the class of the respective sample. Similar to the output of the softmax function, $\boldsymbol{y}$ can be interpreted as a discrete probability distribution over $c$ classes, where all probability mass is concentrated on the true class $j$. The categorical cross-entropy is then defined as the expected negative log-likelihood of the prediction $\hat{\boldsymbol{y}}$ under the discrete data distribution $\boldsymbol{y}$:

$$l_{CE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\mathbb{E}_{\boldsymbol{y}} \left[\log\left(\hat{\boldsymbol{y}}\right)\right] = -\sum_{i=1}^{c} \boldsymbol{y}(i) \log\left(\hat{\boldsymbol{y}}(i)\right). \tag{2.14}$$

The above definition assumes that each sample $\boldsymbol{x}$ gets mapped to one corresponding output $\boldsymbol{y}$. However, some applications require loss computation considering multiple output variables. One prominent example is *image segmentation*, where the task is to assign each pixel of an input image to one of $c$ defined output classes. In such cases, the loss is computed by averaging across all pixels.

### Performance evaluation

Performance evaluation methods enable the comparison of different machine learning methods. In deep learning research, this includes the comparison of network architecture and training strategies (e.g., loss function or optimization technique). Supervised machine learning methods are evaluated based on their prediction performance. Performance is typically assessed on a *test set* $\mathbb{X}^{te} = \left\{\left(\boldsymbol{x}_i^{te}, \boldsymbol{y}_i^{te}\right) \mid 1 \leq i \leq n_{te}\right\}$. Similar to the training dataset, it consists of examples and corresponding labels. The test set is an independent dataset (i.e., it contains samples that were not used for training) and allows to assess the generalizability of a model to unseen data. Assessing the generalizability is crucial to estimate how a model will perform in practical applications.

This section focuses on the evaluation of supervised methods. The lack of labels makes the performance of unsupervised methods (Section 2.2.4) more difficult to quantify. Features learned by unsupervised methods can be clustered and visualized to investigate the learned features. Similarly, training a supervised classifier based on such features can help to assess their performance.

**Confusion matrix** A *confusion matrix* is a tool to analyze the classification performance on a dataset. Given a test dataset, the confusion matrix entry in row $i$ and

column $j$ ($1 \leq i, j \leq c$) specifies how many samples belonging to class $i$ are classified as class $j$:

$$\mathrm{CM}(i,j) = |\left\{ \left(\boldsymbol{x}_k^{te}, \boldsymbol{y}_k^{te}\right) \in \mathbb{X}^{te} \mid \arg\max \boldsymbol{y}_k^{te} = i \wedge \arg\max \hat{\boldsymbol{y}}_k^{te} = j \right\}|. \tag{2.15}$$

The confusion matrix can be used to calculate different performance metrics.

**Metrics** In machine learning, *metric*[9] refers to a function that quantifies the performance of a trained model. The value of a metric allows the comparison of different models (e.g., different architectures or training strategies). They are conceptually similar to a loss function in that they both measure the performance of a model. Compared to a loss function, however, metrics are not optimized during training and do not need to be differentiable. The here discussed metrics for supervised machine learning algorithms quantify the agreement of a model's predictions and the expected labels.

Many metrics can be conveniently computed based on the confusion matrix introduced in the last section: The *accuracy* is defined as the fraction of correctly classified samples. It can be computed from the confusion matrix as the ratio between the sum of diagonal entries and the sum of all entries:

$$\mathrm{accuracy}(\mathrm{CM}) = \frac{\sum_{i=1}^{c} \mathrm{CM}(i,i)}{\sum_{i=1}^{c}\sum_{j=1}^{c} \mathrm{CM}(i,j)} \tag{2.16}$$

The accuracy is easy to interpret, but it needs to be used with care when dealing with imbalanced datasets. In an imbalanced dataset, some classes may occur significantly more frequently than others. A prominent example for this effect can be found in the diagnosis of rare diseases: If the vast majority of patients do not suffer from a specific disease (e.g., 99.9%), a practically useless classifier that always predicts "negative" (i.e., no disease) achieves 99.9% accuracy due to the high class imbalance in the data. In such cases, other metrics are better suited to understand a model's performance.

*Precision* (also referred to as *positive predictive value*) measures the ratio between correctly classified samples from a class $k$ ($1 \leq k \leq c$) and the number of samples predicted as belonging to class $k$:

$$\mathrm{precision}_k(\mathrm{CM}) = \frac{\mathrm{CM}(k,k)}{\sum_{i=1}^{c} \mathrm{CM}(i,k)} \tag{2.17}$$

[9] Metrics in the context of machine learning typically measure a notion of the difference between the predicted and groundtruth values. By that, they are conceptually similar to metrics in the mathematical sense, but they do not necessarily satisfy the mathematical definition of a metric.

*Recall* (also referred to as *true positive rate*) measures the ratio between correctly classified samples from class $k$ and the number of samples belonging to this class:

$$\text{recall}_k(\text{CM}) = \frac{\text{CM}(k,k)}{\sum_{i=1}^{c} \text{CM}(k,i)} \tag{2.18}$$

Precision and recall answer different questions: Precision measures how likely a sample belongs to class $k$ if a model assigns it to class $k$. Recall measures how likely a sample is assigned to class $k$ if it belongs to class $k$. Therefore, precision and recall can give contradicting results (e.g., precision can be high, but recall is low, and vice-versa). Combined metrics can be used if the considered application does not indicate which metric to prefer over another. Combining multiple metrics can be useful in practice, as a single value allows easier comparison of different methods.

The *F1-score* (also referred to as *Dice-score* or *Sørensen-Dice-score*) is computed as the harmonic mean between precision and recall:

$$\text{F1}_k(\text{CM}) = 2\frac{\text{precision}_k(\text{CM})\ \text{recall}_k(\text{CM})}{\text{precision}_k(\text{CM}) + \text{recall}_k(\text{CM})} \tag{2.19}$$

The F1-score has a range of $[0, 1]$. Higher values indicate higher precision and recall and hence better model performance.

Equation 2.19 computes the F1-score on a per-class basis, allowing the performance analysis for specific classes. The *macro F1-score* is obtained by averaging the F1-scores of all classes:

$$\text{F1}(\text{CM}) = \frac{1}{c}\sum_{k=1}^{c} \text{F1}_k(\text{CM})\,. \tag{2.20}$$

Each class contributes equally to the computational of the macro-F1 score, which mitigates the effect of class imbalance in the dataset.

**Ablation studies** The process of training a deep neural network involves many degrees of freedom: The layers defining the architecture of a deep neural network, the optimizer, the learning rate policy, and many other components involved during training can be freely chosen from a large pool of methods. When developing new deep learning methods, it is therefore important to systematically evaluate the influence of newly introduced components or changes made to established training protocols. *Ablation studies* comprise the systematic addition or removal of components (e.g., new layers) to investigate their performance impact.

### 2.2.2 Neural network layers

Deep neural networks are composed of layers. A layer is a differentiable and often parameterized function. It applies a transformation to incoming data and passes it on to the next layer. The layer composition (i.e., their type and how they are connected) determines the architecture of a deep neural network.

The *universal approximation theorem* (Cybenko, 1989) provides an important theoretical foundation for the application of deep neural networks. It states that a deep neural network composed of at least two fully-connected layers (Section 2.2.2) with finite dimensions can approximate any Borel measurable function arbitrarily well. The immediate practical implications of this theorem are limited, as it allows no reasoning about the efficiency of such an approximation. However, it shows that deep neural networks composed of an alternating series of linear and non-linear functions are universal function approximators. This is a crucial insight: It demonstrates that deep neural network constructed from relatively simple "building blocks" (i.e., layers) can approximate complex non-linear functions, which makes them applicable for complex tasks (e.g., image analysis).

The interpretation of layers as basic building blocks for complex model architectures provides an effective way to conceptualize and communicate the structure of deep neural networks. Software frameworks for deep learning (Abadi et al., 2016; Paszke et al., 2019) adopt this concept by allowing users to define deep neural network from pre-defined layers.

The type of data (e.g., vector-valued data, images, graphs) used for a given task determines which types of layers are predominantly used to address the task. The following sections introduce layers that are used in the scope of this work. It introduces general-purpose layers, layers for image data, and layers for graph-structured data.

#### General purpose layers

**Fully-connected layers** A fully-connected layer consists of a linear transformation, followed by a non-linear *activation function*. Given an input vector $\boldsymbol{x} \in \mathbb{R}^{d_i}$, a fully-connected layer is defined as

$$\mathrm{FC}(\boldsymbol{x}; W, \boldsymbol{b}) = \sigma(W\boldsymbol{x} + \boldsymbol{b}), \tag{2.21}$$

where $W \in \mathbb{R}^{d_o \times d_i}$ is a weight matrix, $\boldsymbol{b} \in \mathbb{R}^{d_o}$ is a bias term, and $d_o$ determines the output dimension of the layer. The output dimension is also referred to as the *number of features* that the layer learns.

The non-linear activation function $\sigma$ prevents a series of fully-connected layers from collapsing into a linear transformation. The universal approximation theorem (Cy-

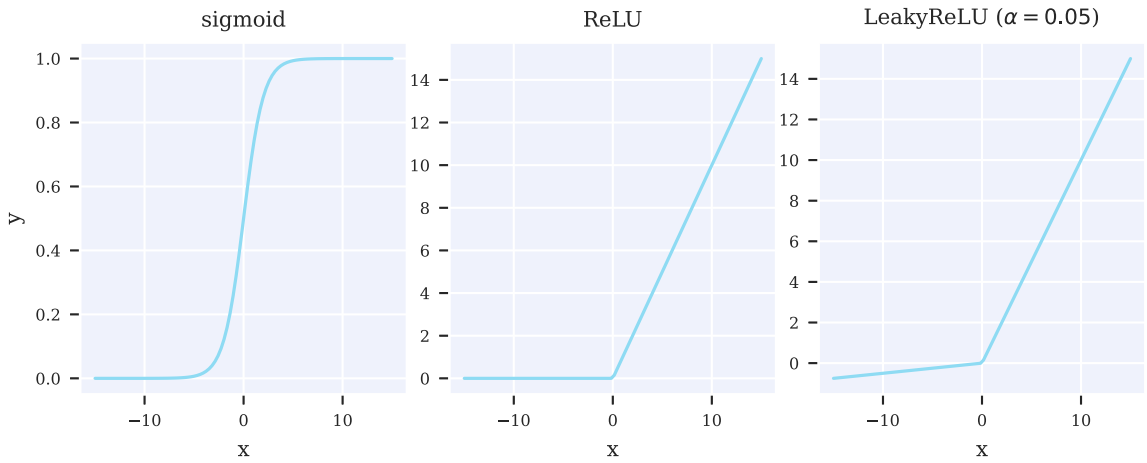



**Figure 2.8:** Activation functions for deep neural networks. The gradient of the *sigmoid* function is close to zero for most inputs, making it prone to suffer from the vanishing gradient problem (Section 2.2.1). The ReLU and leaky ReLU functions have a positive gradient for the entire positive half-line. In addition, the leaky ReLU has small positive gradients for the negative half-line.

benko, 1989) shows that the combination of linear transformations with non-linear activation functions is crucial for the representational capacity of deep neural network.

Fully-connected layers were proposed in the earliest works on deep learning (Rosenblatt, 1958; Fukushima et al., 1982) and have been studied extensively since. Models consisting of multiple fully-connected layers are referred to as multi-layer perceptrons (MLPs).

**Activation functions** An *activation function* (also referred to as *non-linearity*) is a non-linear function. In combination with linear transformations (e.g., in a fully-connected layer), an activation function $\sigma$ breaks the linearity of a deep neural network. This is crucial to enable the learning of complex non-linear relationships in the data (Cybenko, 1989).

The *sigmoid function*

$$\text{sigmoid}(\boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{x})}. \tag{2.22}$$

is one of the earliest practically applied activation functions. It is applied independently to all elements of the input vector $\boldsymbol{x} \in \mathbb{R}^{d_i}$. The gradient of the sigmoid function is close to zero for a wide range of input values (Figure 2.8), which makes it prone to suffer from the vanishing gradient problem (Section 2.2.1).

The *softmax function* (Equation 2.12) is used to normalize a vector. The result can be interpreted as a categorical probability distribution over a set of classes. It is typically used as the final operation in deep neural networks for classification, e.g., as input for the categorical cross-entropy loss (Section 2.2.1).

The rectified linear unit (ReLU) activation function (Hahnloser et al., 2000)

$$\mathrm{ReLU}(\boldsymbol{x}) = \max(0, \boldsymbol{x}) \tag{2.23}$$

is one of the most popular activation function used in modern deep neural network architectures. Compared to the sigmoid function, its gradient is one for the entire positive half-line, making it less prone to suffering from the vanishing gradient problem (Figure 2.8). The *leaky ReLU* activation function is a variant of the ReLU activation function:

$$\mathrm{LeakyReLU}(\boldsymbol{x}) = \max(0, \boldsymbol{x}) + \alpha \min(0, \boldsymbol{x}), \tag{2.24}$$

where $\alpha \in \mathbb{R}^{>0}$ is a positive slope. Leaky ReLU maps negative values to small negative values, which ensures that the gradient of the activation function is always non-zero (Figure 2.8).

**Batch normalization layers** Batch normalization (Ioffe et al., 2015) aims to stabilize the training process to enable faster learning with higher learning rates. The idea of the batch normalization is to normalize the output of each layer before passing it on to the next layer. This ensures that the distribution of input values each layer receives remains approximately fixed during training. Batch normalization has shown to result in more stable and faster training (Ioffe et al., 2015).

Batch normalization computes normalization statistics based on all samples in a batch of training samples. Given a batch of training samples

$$\mathbb{B} = \left\{ \boldsymbol{x}_j \in \mathbb{R}^{d_i} \mid 1 \leq j \leq b \right\} \tag{2.25}$$

with batch size $b = |\mathbb{B}|$, batch normalization is defined as

$$\mathrm{BN}(\boldsymbol{x}; \boldsymbol{\gamma}, \boldsymbol{\beta}) = \boldsymbol{\gamma} \frac{(\boldsymbol{x} - \boldsymbol{\mu}_{BN})}{\boldsymbol{\sigma}_{BN}} + \boldsymbol{\beta}, \tag{2.26}$$

$$\text{with } \boldsymbol{\mu}_{BN} = \frac{1}{b} \sum_{j=1}^{b} \boldsymbol{x}_j, \tag{2.27}$$

$$\boldsymbol{\sigma}^2_{BN} = \frac{1}{b-1} \sum_{j=1}^{b} \left( \boldsymbol{x}_j - \boldsymbol{\mu}_{BN} \right)^2. \tag{2.28}$$

All operations are applied elementwise. The statistics $\boldsymbol{\mu}_{BN}$ and $\boldsymbol{\sigma}_{BN}$ estimate the mean and standard deviation of the samples in the batch $\mathbb{B}$. Mean and standard deviation are used to normalize the data. The parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are trainable, allowing the layer to recover the identity function if needed (by setting $\boldsymbol{\gamma} = \boldsymbol{\sigma}_{BN}$ and $\boldsymbol{\beta} = \boldsymbol{\mu}_{BN}$). When applying batch normalization to high dimensional data (e.g., images), the estimation of mean (Equation 2.27) and standard deviation (Equation 2.28) is performed by averaging over the spatial dimensions of the data (e.g., height and width for images).

One caveat of batch normalization is the dependence on the batch size to obtain estimates of the batch statistics $\boldsymbol{\mu}_{BN}$ and $\boldsymbol{\sigma}_{BN}$. The batch size needs to be sufficiently large to enable reliable estimation of mean and standard deviation. Training with large batch sizes can be computationally expensive, especially when dealing with high-dimensional data like images.

After training, the dependence on the batch composition is undesirable, as results may differ depending on the samples included in a batch. To overcome this limitation, batch normalization behaves differently during inference compared to training. Instead of computing batch statistics on the actual batch, statistics are accumulated during training using an exponential moving average. During inference, these accumulated statistics are then used in place of $\boldsymbol{\mu}_{BN}$ and $\boldsymbol{\sigma}_{BN}$.

Batch normalization is typically applied between the linear operation and the activation function of a layer. For example, a fully-connected layer with batch normalization takes the form

$$\mathrm{FC}_{BN}(\boldsymbol{x}; W, \boldsymbol{\gamma}, \boldsymbol{\beta}) = \sigma(\mathrm{BN}(W\boldsymbol{x}; \boldsymbol{\gamma}, \boldsymbol{\beta}))\,. \tag{2.29}$$

Note that the bias $\boldsymbol{b}$ of the fully-connected layer can be omitted when batch normalization is used, as the bias vector $\boldsymbol{\beta}$ of the batch normalization layer models the same effect.

**Dropout** *Dropout* (Hinton et al., 2012) is a regularization technique to improve the robustness of deep neural networks and to reduce overfitting. It works by setting random entries of an input vector $\boldsymbol{x} \in \mathbb{R}^{d_i}$ to zero, effectively preventing propagation of information through these entries:

$$\mathrm{drop}_i(\boldsymbol{x}; p) = \begin{cases} \boldsymbol{x}(i) & p_i = 0 \\ 0 & p_i = 1 \end{cases}\,. \tag{2.30}$$

The values $p_i \in \{0, 1\}$ are sampled from a Bernoulli distribution with *dropout probability* $p$ and determine which entries are set to zero. Dropout is independently computed for each data sample and training iteration. By randomly disabling parts

of the deep neural network, each training iteration effectively uses a slightly different "subnetwork" of the model. Training with dropout is thus related to ensemble methods (Goodfellow et al., 2016).

Dropout is typically only applied during training. During inference, dropout does not set values to zero, but all output values are scaled by a factor $1 - p$. This correction ensures that the expected output values are identical during training and inference (Hinton et al., 2012).

In addition to its role as a regularization technique, dropout allows uncertainty estimation of deep neural networks (Gal et al., 2016).

**Skip connections** *Skip connections* enable the propagation of information through a deep neural network without passing through all layers of the model. They provide "shortcuts" through a model and have been shown to improve performance (Ronneberger et al., 2015; He et al., 2016a; Huang et al., 2017).

*Residual connections* (He et al., 2016a,b) are used by the ResNet architecture (Section 2.2.3). A residual connection adds the output of a function $h$ (e.g., one or multiple layers) to its input $\boldsymbol{x}$:

$$\text{skip}_{\text{add}}(\boldsymbol{x}) = h(\boldsymbol{x}) + \boldsymbol{x} \tag{2.31}$$

Similarly, the *dense connection* (Huang et al., 2017) of the DenseNet architecture (Section 2.2.3) uses concatenation:

$$\text{skip}_{\text{cat}}(\boldsymbol{x}) = h(\boldsymbol{x}) \,||\, \boldsymbol{x} \tag{2.32}$$

Residual and dense connections skip few layers (i.e., $h$ encompasses one or few layers). Architectures like the U-Net (Section 2.2.3) use long-range skip connections, where $h$ encompasses many layers.

The use of skip connections can stabilize the training behavior of deep neural networks. For example, they can make training less prone to the vanishing gradient problem (Section 2.2.1), as skip connections provide alternative paths for the gradients to "flow" through (He et al., 2016b).

### Layers for image data

This section introduces layers for processing of image data[10]. We model an image $X \in \mathbb{R}^{h \times w}$ with height $h$ and width $w$ as a two-dimensional array[11]. Entries of the array represent the intensity values at the respective pixel positions. We use the function notation to specify the intensity values at certain pixel positions, i.e., $X(i, j) \in \mathbb{R}$ denotes the intensity value of a pixel at row $1 \leq i \leq h$ and column $1 \leq j \leq w$. Following this convention, we model a multi-channel image (e.g., a RGB image) with $c$ channels as a three-dimensional array $X \in \mathbb{R}^{h \times w \times k}$, where $X(i, j, k) \in \mathbb{R}$ specifies the intensity value at row $1 \leq i \leq h$, column $1 \leq j \leq w$, and channel $1 \leq k \leq c$.

In most cases, the output of an image processing layer is itself an image, denoted by $Y$. Depending on the layer, the height, width, and number of channels of the output image can differ from those of the input image.

**Convolutional layers** *Convolutional layers* are the most important layers for processing image data[12]. A convolutional layer applies a discrete convolution of an input image $X$ with a set of $n_K$ *filters* $K = \left\{ K_k \in \mathbb{R}^{u \times u \times c} \,|\, 1 \leq k \leq n_K \right\}$ (also referred to as *kernels*), where $u$ denotes the size of a filter[13] and $c$ denotes the number of channels in an input image $X \in \mathbb{R}^{h \times w \times c}$. Simply speaking, a convolutional layer computes a weighted average across a neighborhood of pixels at pixel position $1 \leq i \leq h$ and $1 \leq j \leq w$, where the weights of the average are trainable parameters specified by the filters $K$:

$$Y(i, j, k) = (X * K)(i, j, k) \tag{2.33}$$

$$= \sum_{i'} \sum_{j'} \sum_{k'} X\big(i - i', j - j', k'\big)\, K_k\Big(1 + i' + \frac{u}{2}, 1 + j' + \frac{u}{2}, k'\Big)\,, \tag{2.34}$$

where $-\frac{u}{2} \leq i', j' \leq +\frac{u}{2}$, $1 \leq k' \leq c$, $1 \leq k \leq n_K$, and $*$ denotes the discrete convolution of an image $X$ with $n_K$ filters. The number of filters is a hyperparameter

[10] All layers presented in this section can be applied to images, as well as any other kind of data that is organized on a regular grid, e.g., 3D volumes. However, for brevity, we restrict our descriptions to the 2D case.

[11] In signal processing, images are typically modelled as functional mappings from a two-dimensional coordinate space to an image space (e.g., intensity values). In this work, we model images as arrays, resembling the conventions of programming languages (e.g., *Python*) or deep learning software frameworks (Abadi et al., 2016; Paszke et al., 2019). This approach allows us to define operations in a way that closely follows their implementation in software.

[12] This section focuses on aspects of convolutional layers that are relevant in the scope of this thesis. Goodfellow et al. (2016) provide a comprehensive overview of convolutional layers and discuss additional variants, properties, and neuroscientific foundations.

[13] While non-square filters are possible, we restrict the definition to square kernels to simplify notation.

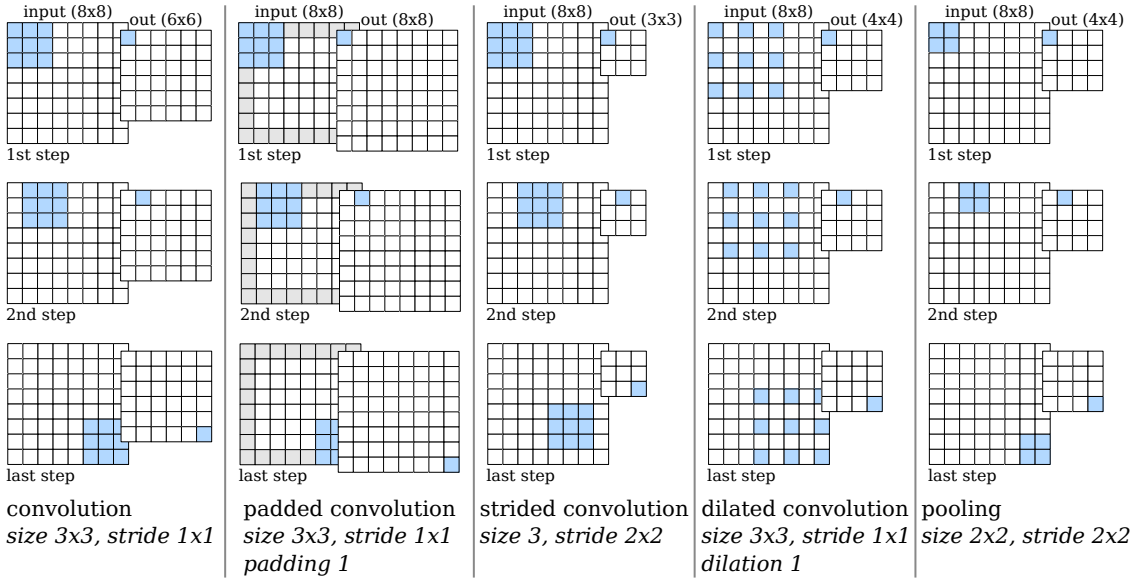



**Figure 2.9:** Illustration of different convolution operations. The first, second, and last steps of each operation are shown to illustrate the underlying principle. Computational dependencies between input and output pixels are shown in blue. Pixels added by padding are shown in gray. An input image of $8 \times 8$ pixels results in differently shaped output images depending on the type of applied convolutional operation. A single-channel input image is shown to improve visualization, but principles are directly applicable to multi-channel images. Note that in this example, strided convolution does not consider all pixels of the input image, which should generally be avoided in practice by proper tuning of image shapes.

of the layer, which specifies the number of channels for $Y$. Similar to hand-crafted linear filters used in image analysis (e.g., the *Sobel filter*, Kanopoulos et al., 1988), each filter can extract specific features from the incoming image. However, compared to hand-crafted filters, the filters $K$ are optimized during the training of a deep neural network. This allows convolutional layers to learn the extraction of task-specific features. Based on the idea that each filter extracts specific features from the incoming image data, the output channels of a convolutional layer are referred to as *feature maps*. Figure 2.9 illustrates the working principles of different types of convolutional operations. A comprehensive overview of convolutional operations, as well as associated arithmetics and visualization, can be found in Dumoulin et al. (2018).

The application of multiple convolutional layers in a row enables the extraction of deep feature hierarchies. By using the feature maps produced by a convolutional layer as input for a second convolutional layer, the second layer can build upon the features extracted by the first layer to extract more complex features. Stacking a series of multiple convolutional layers on top of each other thus enables the extraction of

features with increasing complexity. Architectures constructed this way are referred to as CNNs and have become the state-of-the-art method for many tasks involving images. It has been shown that early layers (i.e., close to the input) of CNNs tend to extract primitive features like edges, while later layers tend to learn increasingly more abstract features like corners or parts of objects (Goodfellow et al., 2016).

Equation 2.34 assumes that the output image $Y$ has the same spatial dimensions $h$ and $w$ as the input image $X$. However, this is only true when *padding* is applied. Padding artificially increases the spatial dimensions of the input image $X$ before performing the convolution operation to prevent issues at the borders of the image, where some pixels "outside" the input image (i.e., $i < 1$, $i > h$, $j < 1$ and/or $j > w$) are required for the computation. Pixels added by padding are typically chosen to be constant (e.g., zero-padding) or are created by mirroring or replicating the border pixels. Padding has the advantage of keeping the spatial size of images constant when applying a series of convolutional layers, which is crucial to building deep network architectures. However, padding can also introduce artifacts at the borders of feature maps, which is particularly problematic in image segmentation tasks. In such cases, padding can be omitted, resulting in a (typically small) reduction of the feature map size with each convolutional layer.

The convolutional operator in Equation 2.34 applies a discrete convolution at *each* position in the image. As described previously, this operation does not reduce the size of the image when padding is used, or only slightly if no padding is used. Applying a series of convolutional layers in a CNN can thus become computationally expensive, especially for large images. *Strided convolution* overcomes this issue by computing convolutions on a coarser grid. It can be defined as

$$(X *_s K)(i_s, j_s, k) = (X * K)(s \cdot i_s, s \cdot j_s, k) \tag{2.35}$$

with $1 \leq i_s \leq \frac{h}{s}$, $1 \leq j_s \leq \frac{w}{s}$, $1 \leq k \leq n_K$, and $s \in \mathbb{N}$ denoting the stride parameter. Consequently, a convolution with stride $s$ reduces the height and width of the input image by the factor $\frac{1}{s}$, which can reduce computational and memory requirements of subsequent computations.

*Dilated convolutions*[14] introduce a dilation rate $d \in \mathbb{N}$, which defines a spacing (or "gaps") between the entries of a convolutional kernel. This "inflation" step widens the filters without increasing the number of parameters, enabling dilated convolutional layers to capture more context in the input. Dilated convolution with filter size $u$ and dilation rate $d$ is equivalent to performing regular convolution with filters

[14] Dilated convolutions are sometimes also referred to as *atrous convolutions*, derived from the French term "à trous", which means "with holes".

$K_k^d \in \mathbb{R}^{u_d \times u_d \times c}$ of size $u_d = u + (u-1)(d-1)$ with[15]

$$K_k^d(i,j,k') = \begin{cases} K_k\left(1 + \frac{i-1}{d}, 1 + \frac{j-1}{d}, k'\right) & d \mid (i-1) \wedge d \mid (j-1) \\ 0 & \text{otherwise} \end{cases} \tag{2.36}$$

with $1 \leq i, j \leq u_d$ and $1 \leq k' \leq c$. For efficiency, practical implementations do not perform multiplication with zero entries in the filter. Regular convolutional can be regarded as dilated convolution with $d = 1$. Dilated convolutions can be combined with strided convolutions.

Convolutions are linear operations. Similar to fully-connected layers, they are combined with non-linear activation functions (Section 2.2.2) to model non-linear relationships. A convolutional layer takes the form

$$\text{conv}(X; K, \boldsymbol{b}) = \sigma(X * K + \boldsymbol{b}), \tag{2.37}$$

where $\sigma$ is an activation function (e.g., ReLU) and $\boldsymbol{b} \in \mathbb{R}^{n_K}$ is a vector containing one bias term per feature map. $K$ and $\boldsymbol{b}$ are trainable parameters of the convolutional layer.

The number of parameters of a convolutional layer is solely determined by the filter size $u$ and the number of filters per kernel $n_K$. Thus, the parameter count is independent of the number of input values (e.g., pixels). This property has made convolutional layers the predominant layer type for image processing. In comparison, the number of parameters of a fully-connected layer depends on the number of input values. This dependence makes fully-connected layers impractical for image processing, as the number of pixels grows quickly with the height and width of an image. This issue becomes even more severe for higher-dimensional data like 3D volumes. Convolutional layers manage to keep the number of parameters independent of the input size by leveraging the concept of *parameter sharing*. It describes the idea of reusing the same set of parameters across multiple locations in the input image. The underlying assumption is that if a filter can extract a certain feature (e.g., an edge), this extraction step is independent of the specific location in the image and can be applied at arbitrary locations.

Convolutional layers further leverage the concept of *sparse interactions* to limit the number of parameters. The underlying assumption of sparse interaction is that meaningful features can be extracted by considering only a small set of input pixels at each location. As a result, filters in convolutional layers are typically small (e.g., $3 \times 3$ pixels) and capture only local features. More complex features can then be extracted by stacking multiple convolutional layers on top of each other, with each layer extracting more complex features than the previous one.

[15] $a \mid b$ denotes that $a$ divides $b$, i.e., $a \mid b \Leftrightarrow b \mod a = 0$.

**Pooling layers** *Pooling layers* reduce the size of images and feature maps. Reducing the size of images and feature maps is required to make processing large images computationally feasible. In addition, condensing extracted features allows to create low dimensional representations of input images.

Similar to convolutional layers, pooling layers aggregate a small subset of pixels into a single pixel (Figure 2.9). However, pooling layers typically do not have trainable parameters and do not contribute to the total number of parameters of a deep neural network.

A pooling layer is generally defined as

$$\text{pool}(X)\big(i_p, j_p, k\big) = \text{agg}\left(\left\{X\big(p \cdot i_p + i, p \cdot j_p + j, k\big) \mid 1 - p \leq i, j \leq 0\right\}\right), \quad (2.38)$$

with $1 \leq i_p \leq \frac{h}{p}$ and $1 \leq j_p \leq \frac{w}{p}$. The pool size $p \in \mathbb{N}$ determines how many pixels are aggregated at each position and consequently the factor by which the image is resized. Typical choices for the aggregation function agg are maximum (*max-pooling*) or mean (*average-pooling*).

Global average pooling (GAP) is a special kind of pooling: Here, the pool size $p$ in each dimension is set equal to the height $h$ and width $w$ of the incoming image. This effectively removes the spatial dimension of the image, resulting in a vector with $k$ entries. The aggregation of the spatial dimensions significantly reduces the number of values to be processed, making it feasible to process the output of GAP using a fully-connected layer. GAP is typically applied in CNN architectures for image classification, which consist of several convolutional layers for feature extraction, followed by GAP to reduce the extracted feature maps into a vector, and finally one or more fully-connected layers to perform the classification.

### Layers for graph data

Many real-world tasks can be modeled using graphs, including combinatorial optimization (e.g., the *traveling salesman problem*), shortest path computation (Dijkstra, 1959), protein-protein interaction prediction (Zitnik et al., 2017), and social network analysis (Zachary, 1977). In Chapter 7, we model brain mapping as a node classification problem in a graph representing the brain surface.

A graph is defined[16] as a tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_i \mid 1 \leq i \leq n_\mathcal{V}\}$ describes a set of $n_\mathcal{V}$ *nodes* (or *vertices*) and $\mathcal{E}$ is a set of edges connecting nodes. An edge going from node $u \in \mathcal{V}$ to $v \in \mathcal{V}$ is denoted as $(u, v) \in \mathcal{E}$. A graph with $(u, v) \in \mathcal{E} \Leftrightarrow (v, u) \in \mathcal{E}$ is called *undirected*. The *neighborhood* of a node $v_i$ is defined as the set of all nodes that are directly connected to the node, i.e., $\mathcal{N}(i) = \left\{v_j \in \mathcal{V} \mid \big(v_i, v_j\big) \in \mathcal{E}\right\}$. The number of neighbors $\deg(v_i) = |\mathcal{N}(i)|$ of a node $v_i$ is referred to as the *degree*

[16] Our notation for graphs largely follows the notation used in Hamilton (2020).

of a node. Going further, the *k-hop neighborhood* of a node $v_i$ contains all nodes that can be reached from $v_i$ by travelling along at most $k$ edges (*hops*). It can be recursively defined as

$$\mathcal{N}_k(i) = \mathcal{N}(i) \cup \bigcup_{v_j \in \mathcal{N}(i)} \left(\mathcal{N}_{k-1}(j) \setminus \{v_i\}\right), \tag{2.39}$$

with $\mathcal{N}_1(i) \equiv \mathcal{N}(i)$.

The connectivity between nodes can be represented using an *adjacency matrix* $A \in \{0,1\}^{n_\mathcal{V} \times n_\mathcal{V}}$, with $A(i,j) = 1 \Leftrightarrow \left(v_i, v_j\right) \in \mathcal{E}$ and $A(i,j) = 0 \Leftrightarrow \left(v_i, v_j\right) \notin \mathcal{E}$. The adjacency matrix of an undirected graph is symmetric.

Nodes and/or edges of a graph can have *features* (also referred to as *attributes*). Node features[17] are denoted as $\mathcal{F}_\mathcal{V} = \left\{\boldsymbol{x}_{v_i} \,|\, v_i \in \mathcal{V}\right\}$, where $\boldsymbol{x}_{v_i} \in \mathbb{R}^{d_i}$ describes the feature vector associated with node $v_i$.

**Graph neural networks** Graph neural networks (GNNs) (Scarselli et al., 2008) are a class of deep neural networks for processing graph data[18]. Fully-connected layers have no direct way to incorporate the neighborhood information encoded in a graph, and convolutional layers can only operate on data defined on a regular grid[19]. In comparison, GNNs take neighborhood relations, as well as node and edge features into account.

Graph convolutional networks (GCNs) are a group of GNN architectures that generalize the idea of convolutional layers to the arbitrary neighborhood structure of graphs. GCNs can be grouped into two categories: *spectral* and *spatial* GCNs.

Spectral GCNs (Bruna et al., 2014; Defferrard et al., 2016) compute an eigenbasis from the adjacency matrix of a given graph to transform it into the spectral domain. Convolutions are then implemented as a multiplication in the spectral domain. One drawback of spectral GCNs is that the eigenbasis of graphs used during training and inference must be identical. As the eigenbasis is determined by the adjacency matrix, which encodes the topology of the graph, spectral GCN can only be applied to graphs that share the same topology as the training graph. This property makes spectral GCNs a *transductive* approach. In addition, the eigenvalue decomposition performed by GCNs can be computationally expensive. Approximations (Defferrard et al., 2016)

[17] We do not introduce notation for edge features because they are not used in the scope of this work. Also, while the definition depicts the node features to be vectors, any kind of data (e.g., images) could technically be used as node features.

[18] Hamilton (2020) provides a comprehensive overview of GNNs and other methods for machine learning on graphs. This section focuses on aspects that are most relevant in the scope of this thesis.

[19] Note that images can be regarded as graphs with a specific topology on a regular grid.

can reduce the computational requirements, but the application of spectral GCNs to large graphs remains challenging.

Spatial GCNs (Hamilton et al., 2017; Kipf et al., 2017; Veličković et al., 2018) operate directly in the graph domain. As a result, spatial GCNs are not restricted to graphs of a specific topology. This property makes them an *inductive* approach. They can be applied to new graphs during inference.

The layers of a spatial GCN can be modelled using the *message passing* (Gilmer et al., 2017) framework:

$$\mathrm{mp}\big(\boldsymbol{x}_{v_i}; \theta_{msg}, \theta_{upd}\big) = \mathrm{upd}\big(\boldsymbol{x}_{v_i}, \mathrm{msg}\big(\boldsymbol{x}_{v_i}; \theta_{msg}\big); \theta_{upd}\big)\,, \tag{2.40}$$

$$\text{with } \mathrm{msg}\big(\boldsymbol{x}_{v_i}; \theta_{msg}\big) = \mathrm{agg}\Big(\big\{\boldsymbol{x}_{v_i}\big\} \cup \Big\{\boldsymbol{x}_{v_j} \,|\, v_j \in \mathcal{N}(i)\Big\}; \theta_{msg}\Big)\,. \tag{2.41}$$

The message passing framework iteratively updates the features of all nodes $v_i \in \mathcal{V}$ in a graph $\mathcal{G}$ based on their respective neighborhood. A message passing layer comprises two steps: *message* creation and node *update*[20]. A message $\mathrm{msg}\big(\boldsymbol{x}_{v_i}; \theta_{msg}\big)$ is created by aggregating the features of all neighbors $\mathcal{N}(i)$ of a node $v_i$. The aggregation function "agg" (parameterized by $\theta_{msg}$) is required to be invariant to permutations of the input, as no specific order can be imposed on the set of nodes. The created message is used to update the features of $v_i$ using an update function "upd", which is parameterized by $\theta_{upd}$. The aggregation function and the update function are typically implemented using general-purpose neural network layers (e.g., fully-connected layer layers). Thus, they can be optimized using the backpropagation algorithm (Section 2.2.1).

One message passing layer (i.e., one update step) updates the features of all nodes based on their direct neighborhoods. By stacking $k$ message passing layers on top of each other, a GCN can compute features based on the $k$-hop neighborhood $\mathcal{N}_k(i)$ of $v_i$. A $k$-layer GCN recursively computes an output graph[21]

$$\mathcal{G}^{(k)} = \Big(\mathcal{V}, \mathcal{E}, \mathcal{F}_{\mathcal{V}}^{(k)}\Big)\,, \tag{2.42}$$

$$\text{with } \mathcal{F}_{\mathcal{V}}^{(k)} = \Big\{\mathrm{mp}_k\Big(\boldsymbol{x}_{v_i}^{(k-1)}; \theta_{msg}^{(k)}, \theta_{upd}^{(k)}\Big) \,|\, \boldsymbol{x}_{v_i}^{(k-1)} \in \mathcal{F}_{\mathcal{V}}^{(k-1)}\Big\}\,, \tag{2.43}$$

where $\mathcal{G}^{(0)} = \Big(\mathcal{V}, \mathcal{E}, \mathcal{F}_{\mathcal{V}}^{(0)}\Big)$ is an input graph with nodes $\mathcal{V}$, edges $\mathcal{E}$, and node features $\mathcal{F}_{\mathcal{V}}^{(0)} = \Big\{\boldsymbol{x}_{v_i}^{(0)} \,|\, v_i \in \mathcal{V}\Big\}$.

[20] Software frameworks for training of GNNs enable the definition of GNN layers by implementing functions for message creation and node update (Fey et al., 2019).

[21] We only consider GCNs that leave the graph topology (i.e., nodes and edges) unchanged. Depending on the task, GCNs can alter the topology of a graph (e.g., for graph generation or graph reconstruction).

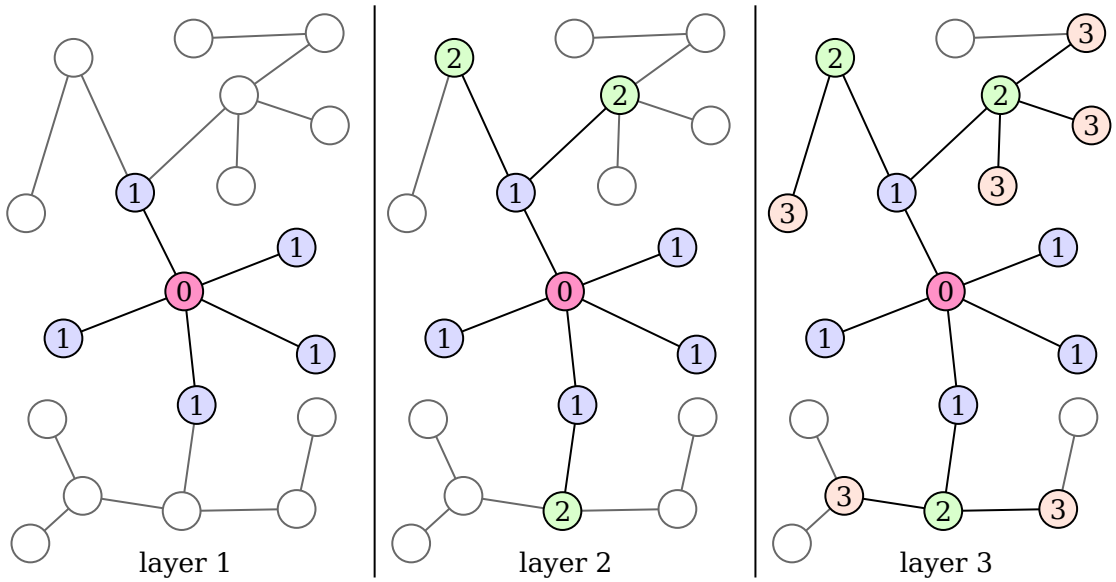



**Figure 2.10:** Illustration of GCN feature aggregation in an undirected graph. The output features of a selected node (red) are computed by iteratively including a larger neighborhood around the node. The numbers inside the nodes indicate how many GCN layers need to be applied to incorporate features of the respective nodes for feature computation of the selected node. White nodes are not considered for feature computation at the respective layer.

Figure 2.10 illustrates the message propagation in a GNN with three layers (i.e., three message passing layers): The first layer updates the features of a selected node (red) based on its immediate neighborhood (blue). After the second layer, the features of the selected node are effectively computed based on its 2-hop neighborhood (blue and green nodes). The third layer repeats this process to effectively incorporate features from the 3-hop neighborhood (blue, green, and orange nodes).

Training a GCN is performed largely analogous to training other deep neural networks. To create a batch of $b$ samples for a training iteration (e.g., Equation 2.3), a subgraph of the original graph is created. Given a batch $\mathbb{B}_\mathcal{V} \subset \mathcal{V}$ of nodes, a subgraph

$$\begin{aligned} \mathcal{G}_\mathbb{B} &= \left(\mathcal{V}_\mathbb{B}, \mathcal{E}_\mathbb{B}, \mathcal{F}_{\mathcal{V}_\mathbb{B}}\right) \\ \mathcal{V}_\mathbb{B} &= \mathbb{B}_\mathcal{V} \cup \bigcup_{v_i \in \mathbb{B}_\mathcal{V}} \mathcal{N}_k(i) \\ \mathcal{E}_\mathbb{B} &= \{(u,v) \in \mathcal{E} \mid u \in \mathcal{V}_\mathbb{B} \wedge v \in \mathcal{V}_\mathbb{B}\} \end{aligned} \tag{2.44}$$

is created, which contains all nodes and edges required for computing output features for $\mathbb{B}_\mathcal{V}$ using a $k$-layer GCN.

GCNs can be applied to address different tasks. In *node classification tasks*, a GCN aims to classify the individual nodes of a graph. In comparison, a GCN for

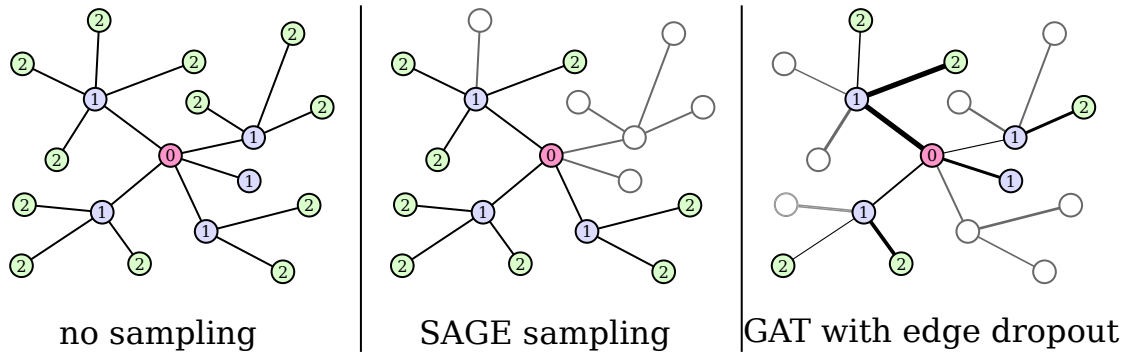



**Figure 2.11:** Subgraph sampling methods for GNN training. Without subsampling, the entire $k$-hop neighborhood of a selected node (red) is included in the sampled subgraph. SAGE samples a fixed number (here three) of neighbors per node, which limits the computational requirements and improves robustness to varying graphs. GAT assigns trainable weights (i.e., *attention*) to neighboring nodes, indicated by edges with varying width. It additionally applies dropout to the learned attention coefficients, resulting in a stochastic neighborhood sampling similar to that employed by SAGE.

*graph classification* analyzes the structure and features of an entire graph to assign a single label to the graph as a whole. Other tasks involve *graph generation*, *graph representation learning*, and *graph reconstruction* (Wu et al., 2020). In this work, we focus on node classification tasks.

Layers used in GCNs differ in how they implement aggregation and update in the message passing framework. The following sections introduce layers used in the scope of this work.

**SAGE** SAGE (SAmple and aggreGatE) (Hamilton et al., 2017) is one of the first methods employing the subgraph sampling scheme described in Equation 2.44, enabling its application to large graphs. SAGE further introduces a stochastic neighborhood sampling procedure (Figure 2.11). The stochastic neighborhood sampling creates batches $\mathcal{G}_{\mathbb{B}}$ by sampling fixed-size neighborhoods around each node rather than using the full neighborhood $\mathcal{N}(i)$. The subsampling ensures that the degree of nodes in the constructed subgraph does not exceed a predefined upper bound. This approach prevents the creation of large subgraphs in cases where $\mathcal{G}$ is densely connected (i.e., has a high average node degree) and limits the computational and memory requirements.

Hamilton et al. (2017) evaluate multiple possible aggregations functions to be used in the message passing framework, all of which follow a similar pattern. SAGE with

mean aggregation takes the following form:

$$\begin{aligned} \mathrm{agg}_{\mathrm{SAGE}}\big(\boldsymbol{x}_{v_i}\big) &= \frac{1}{|\mathcal{N}(i)|} \sum_{v_j \in \mathcal{N}(i)} \boldsymbol{x}_{v_j} \\ \mathrm{upd}_{\mathrm{SAGE}}\big(\boldsymbol{x}_{v_i}; W, \boldsymbol{b}\big) &= \mathrm{FC}\big(\boldsymbol{x}_{v_i} || \, \mathrm{agg}_{\mathrm{SAGE}}\big(\boldsymbol{x}_{v_i}\big); W, \boldsymbol{b}\big) \end{aligned} \tag{2.45}$$

Messages are constructed by averaging features of neighboring nodes. Averaged features are then concatenated with the node features $\boldsymbol{x}_{v_i}$ and passed through a fully-connected layer with $W \in \mathbb{R}^{d_o \times 2d_i}$ and $\boldsymbol{b} \in \mathbb{R}^{d_o}$. The concatenation operation serves as a skip-connection (Section 2.2.2), which enables efficient propagation of information through the network and improves performance (Hamilton et al., 2017).

**Graph Attention Network (GAT)** The mean aggregation used in SAGE assigns equal weights to all neighbors of a node. However, *attention mechanism* have been shown to improve the performance of deep neural networks in a variety of settings (Vaswani et al., 2017; Dosovitskiy et al., 2020). Attention adaptively assigns different weights to different samples of the input, enabling a neural network layer to "attend" to certain samples more than to others (Figure 2.11). Graph attention networks (GATs) (Veličković et al., 2018; Brody et al., 2021) apply the idea of attention to GCNs. They compute a weighted average

$$\begin{aligned} \mathrm{agg}_{\mathrm{GAT}}\big(\boldsymbol{x}_{v_i}\big) &= \frac{1}{|\mathcal{N}(i)|} \sum_{v_j \in \mathcal{N}(i)} \alpha_{ij} \boldsymbol{x}_{v_j} \\ \mathrm{upd}_{\mathrm{GAT}}\big(\boldsymbol{x}_{v_i}; W, \boldsymbol{b}\big) &= \mathrm{FC}\big(\boldsymbol{x}_{v_i} || \, \mathrm{agg}_{\mathrm{GAT}}\big(\boldsymbol{x}_{v_i}\big); W, \boldsymbol{b}\big) \end{aligned} \tag{2.46}$$

with

$$\alpha_{ij} = \frac{\exp\Big(e\Big(\boldsymbol{x}_{v_i}, \boldsymbol{x}_{v_j}\Big)\Big)}{\sum_{v_k \in \mathcal{N}(i)} \exp\big(e\big(\boldsymbol{x}_{v_i}, \boldsymbol{x}_{v_k}\big)\big)}, \tag{2.47}$$

$$e\Big(\boldsymbol{x}_{v_i}, \boldsymbol{x}_{v_j}\Big) = \boldsymbol{a}^\top \mathrm{LeakyReLU}\Big(W \Big(\boldsymbol{x}_{v_i} || \boldsymbol{x}_{v_j}\Big)\Big), \tag{2.48}$$

$W \in \mathbb{R}^{d_o \times 2d_i}$, and $\boldsymbol{b} \in \mathbb{R}^{d_o}$. To compute the output features for a node $v_i$, a GAT layer computes attention coefficients $\alpha_{ij} \in [0, 1]$ for each neighbor $v_j \in \mathcal{N}(i)$ of $v_i$. The computed attention coefficients determine how much attention the layer attributes to each neighbor. The vector $\boldsymbol{a} \in \mathbb{R}^{d_o}$ is a trainable parameter.

Veličković et al. (2018) propose to apply dropout (Section 2.2.2) to the attention coefficients $\alpha_{ij}$ during training (Figure 2.11). Thus, features are computed based on stochastic neighborhoods. This approach has a similar effect as the sampling procedure of SAGE.

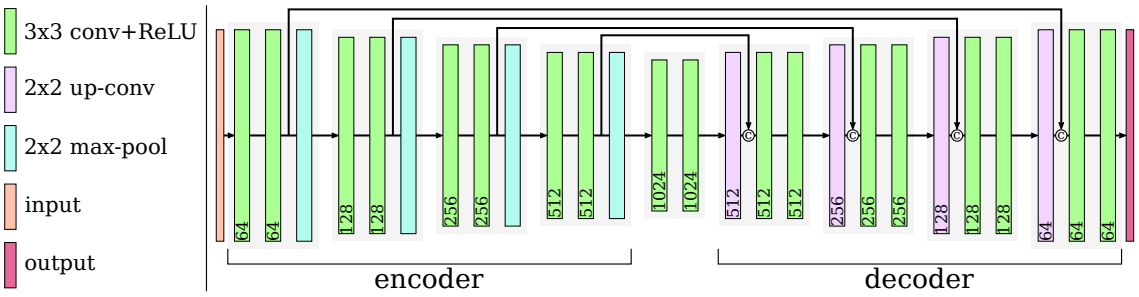



**Figure 2.12:** U-Net architecture for semantic image segmentation proposed by Ronneberger et al. (2015). The model is composed of an *encoder* comprised of convolutional layers and pooling layers, and a *decoder* comprised of convolutional layers and upsampling operations. The encoder extracts features with decreasing spatial resolutions from the incoming image. The decoder increases the spatial resolution to create the final segmentation output. Long-range skip-connections facilitate efficient information propagation and allow the model to combine detailed features in the encoder with complex features in the decoder through concatenation. The numbers in the blocks denote the filter count in the respective convolutional layer. The encircled "c" denotes concatenation along the feature dimension.

### 2.2.3 Neural network architectures

The layer composition of a deep neural network (i.e., type, number, and connection of layers) defines its architecture. The following sections introduce deep neural network architectures that provide the basis for the architectures used in this work.

**Documenting neural network architectures** Documenting deep neural network architectures is a crucial aspect of deep learning research (e.g., in scientific publications). Textual descriptions of an architecture (e.g., filter size, number of filters) can be difficult to understand, especially for architectures with complex connectivity (e.g., using skip-connections). Block diagrams allow documenting model architectures concisely and completely. In a block diagram, each layer (or a composition of multiple layers) is represented as a block. Blocks are annotated with text describing the parameters of layers. Connections between layers (i.e., which layer receives input from which layer) are visualized using arrows.

We use block diagrams to visualize and describe different architectures. An example for a block diagram is shown in Figure 2.12.

#### U-Net

The U-Net (Ronneberger et al., 2015) is an *encoder-decoder* architecture (Figure 2.12) for image segmentation tasks, where the task is to assign all image pixels to the corresponding class. The *encoder* part of the model receives an input image and passes

it through a series of convolutional layers (Section 2.2.2), interleaved with pooling layers (Section 2.2.2) to reduce the spatial resolution of feature maps. Consecutive convolutional layers with identical parameterization (e.g., same number of filters) are often visualized as a block (Figure 2.12).

The result of the encoder is passed on to the *decoder*. The decoder consists of convolutional layers, interleaved with upsampling operations to increase the spatial resolutions of feature maps. The upsampling is implemented by image resizing (e.g., nearest-neighbor), followed by a convolutional layer.

The encoder-decoder design leads to a loss of detail in the extracted feature maps, which hinders the creation of detailed segmentations. The U-Net employs long-range skip-connections (Section 2.2.2) between the encoder and the decoder to mitigate this effect. Feature maps from the encoder are passed over and concatenated to feature maps of the decoder. The skip-connections allow the U-Net to combine fine image details available in the encoder with complex feature representations in the decoder, which ultimately leads to high-resolution segmentations.

The name "U-Net" originates from the encoder-decoder architecture: The encoder reduces the spatial resolution (i.e., the downward path), while the decoder increases it again (i.e., the upward path).

### Residual networks

The depth of a deep neural network determines its ability to learn complex feature hierarchies, which is crucial for image classification tasks (He et al., 2016a). However, He et al. (2016a) observed that deep architectures often obtain lower classification performance than shallower models, even though deeper models should theoretically be able to recover the behavior of shallower models by learning the identity function in some layers. This *degradation problem* limits the use of model architectures with many layers.

He et al. (2016a,b) propose *residual connections* to address the degradation problem. The resulting family of architectures is called residual networks (ResNets). ResNets were the first architecture type that allowed training deep neural networks with many (e.g., 50, 101, 151, 200) layers and significantly improved the performance for image classification tasks.

A residual connection is a skip-connection (Section 2.2.2) that adds the input of a layer to its output:

$$g(\boldsymbol{x}) = x + f(\boldsymbol{x}) \tag{2.49}$$

$$\Leftrightarrow g(\boldsymbol{x}) - x = f(\boldsymbol{x}) \tag{2.50}$$

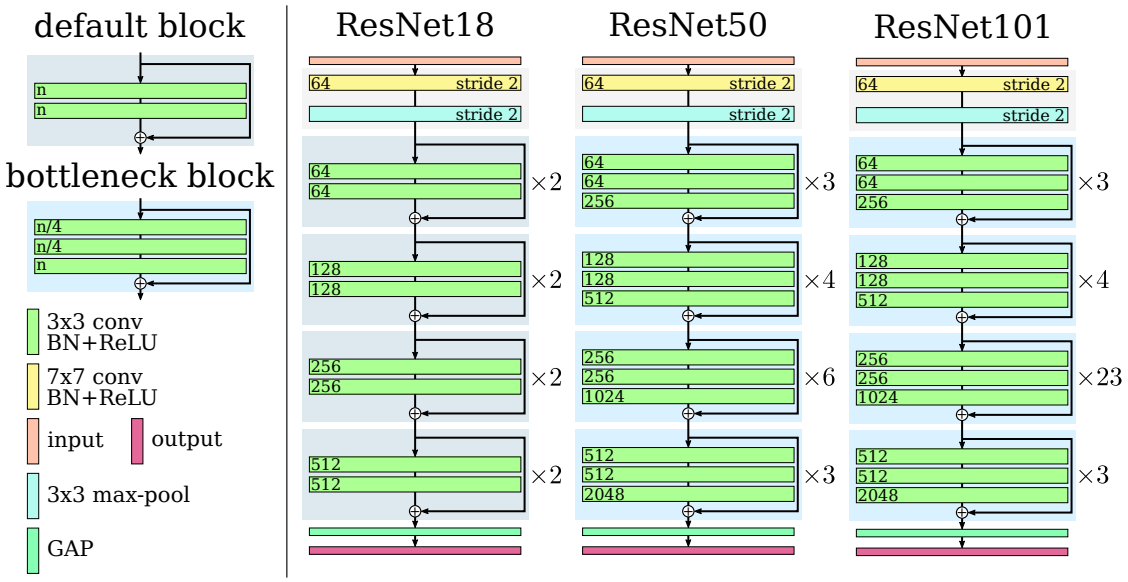



**Figure 2.13:** ResNet architectures (He et al., 2016a) used in this work. Each model consists of a downsampling block, which reduces the size of input images by strided convolutions. It follows a series of residual blocks. `ResNet18` uses *default residual blocks* comprising two convolutional layers with a residual connection. `ResNet50` and `ResNet101` use *bottleneck residual blocks* comprising three convolutional layers with a residual connection. The output of the model is obtained by reducing the output of the residual blocks using GAP. The resulting vector can be used for classification or other tasks. When the number of convolutional filters changes between two blocks, the first convolutional layer of the second block uses a stride of two to reduce the feature map dimensions. Numbers in the blocks denote the number of filters in the respective convolutional layer. The encircled "+" denotes addition.

Here, $g(\boldsymbol{x})$ is a *residual layer* with input $\boldsymbol{x}$, and $f(\boldsymbol{x})$ comprises one or multiple neural network layers. As shown in Equation 2.50, the layer $f(\boldsymbol{x})$ learns the residual between the input and the output of the residual layer. Residual layers can approximate the identity function by driving the weights of $f(\boldsymbol{x})$ towards zero, which is easier than approximating the identity mapping directly (He et al., 2016a). This property enables the efficient training of deeper models and mitigates the degradation problem. The skip-connections also function as "shortcuts" between layers and can improve the stability of the gradient propagation (He et al., 2016b).

He et al. (2016a) propose different model architectures based on residual connections. Figure 2.13 visualizes three ResNet architectures that are used in this work: `ResNet18`, `ResNet50`, and `ResNet101`. The number in the name specifies the number of layers that make up the model. All models start with a common downsampling block, which reduces the size of input images by strided convolutions. After the downscaling follows a series of residual blocks. `ResNet18` uses *default residual blocks*,

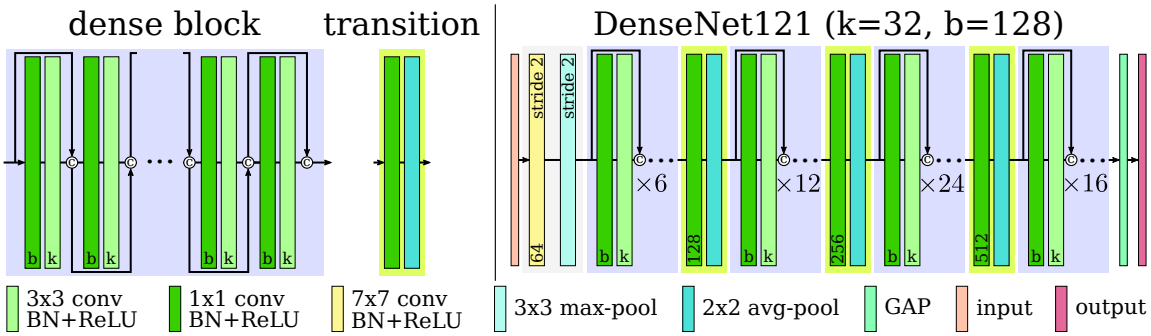



**Figure 2.14:** DenseNet architecture (Huang et al., 2017) used in this work. A DenseNet is composed of *dense blocks* and *transition blocks*. Each layer in a dense block receives the output of all preceding layers in the same block as input. Transition blocks reduce the number and spatial dimension of feature maps. Numbers in the blocks denote the number of filters in the respective convolutional layer. The encircled "c" denotes concatenation along the feature dimension.

each comprising two convolutional layers with a residual connection. `ResNet50` and `ResNet101` use *bottleneck residual blocks*, which consist of three convolutional layers with a residual connection, where the first two convolutional layers use a reduced number of filters. When the number of convolutional filters changes between blocks, the first convolutional layer after the change uses a stride of two to reduce the spatial dimension of the feature maps. Finally, GAP reduces the output of residual blocks into a vector, which can then be used for classification or other tasks. Following He et al. (2016b), the ResNets used in this work use *pre-activated residual connections*, which means that residual connections originate between the convolutional operation and batch normalization.

### Densely connected networks

The densely connected network (DenseNet) (Huang et al., 2017) is an architecture for image analysis tasks. A DenseNet employs *dense blocks* composed of a series of convolutional layers, where each layer receives the output of *all* preceding layers in the same block as input. The layers of each block are therefore *densely connected*. The intuition behind this design is that each new layer can focus on extracting new features from the incoming data, without the need to "remember" and retain relevant input features.

Figure 2.14 illustrates the `DenseNet121` architecture proposed by Huang et al. (2017), which is used in this work. It is comprised of four dense blocks, interleaved with *transition blocks*. A dense block is defined by a *growth rate* $k$ and a *bottleneck size* $b$. In a dense block, feature maps are repeatedly passed through bottleneck layers with $b$ filters and feature extraction layers with $k$ filters, where the input for each

layer comprises the output of all preceding layers in the block. For `DenseNet121`, the growth rate is $k = 32$ and the bottleneck size is $b = 4k = 128$. The transition blocks reduce the number of features in the feature maps and reduce their spatial dimensions using average pooling. The final output is obtained by using GAP to reduce the output of the last dense block into a vector.

#### Graph neural networks

GNN architectures are constructed by stacking multiple GCN layers. This is possible since the result of a GCN operation is itself a graph, which can be processed by following layers (Equation 2.42). While it is possible to mix different kinds of GCN layers within a single model (e.g., SAGE and GAT layers), it is common practice to construct GNN architectures from only one type of layer (You et al., 2020). Each layer can use a different parameterization, for example, to change the dimension of output features. It can further be beneficial to add pre-processing or post-processing layers (You et al., 2020). Pre- and post-processing layers are typically implemented using fully-connected layers, which do not take the graph structure into account, but can be useful to prepare features for following GCN layers, or to map the result of several GCN layers to a task-specific output space (e.g., for classification).

Compared to the field of image processing, for which strong baseline architectures (e.g., `ResNet50`) have been established in recent years, the field of graph processing has not yet established such standard architectures. While there have been some efforts (You et al., 2020) to systematically study the design space of GNNs, it remains necessary to identify a suitable GNN architecture for a specific task.

### 2.2.4 Beyond supervised machine learning

Supervised machine learning enables the training of models for specific tasks. However, the costs for labeling the required training data can be significant. This gives rise to alternative approaches that require much less or no labeling, but are still directly or indirectly useful for solving certain practical tasks.

*Unsupervised learning* methods do not require any labels, making them attractive for the analysis of large, unlabeled datasets that can be acquired inexpensively (e.g., by scraping the web for images). Examples for unsupervised learning involve clustering algorithms (e.g., *k-means clustering* (Lloyd, 1982) or *hierarchical clustering* (Ward Jr, 1963)) or principal component analysis (PCA) (Pearson, 1901). While unsupervised approaches can uncover semantically meaningful structures in the data, they typically require further processing steps to make them useable for practical applications (e.g., assigning meaningful labels to clusters or training a supervised classifier on top of principal component projections).

The transition between supervised and unsupervised methods is not strict. *Semi-supervised* learning describes methods that work with datasets consisting of a small number of labeled examples and a large number of unlabeled examples. Semi-supervised methods belong to a larger group of *weakly supervised* methods, which can work with incomplete, noisy, or imprecise labels.

Self-supervised learning (SSL) is a special case of semi-supervised learning (Jing et al., 2019). Instead of optimizing a deep neural network to directly solve a given *downstream task* (i.e., a task that has practical relevance) using supervised learning, a model is instead tasked to solve an *auxiliary task* (also referred to as *pretext task*). An auxiliary task is a task for which labels can be easily acquired, and that can only be solved by learning concepts that are also useful for solving the downstream task. Auxiliary tasks are constructed by exploiting known structures in the data. Jigsaw puzzles (Noroozi et al., 2016), geometric transformation prediction (Gidaris et al., 2018) and context prediction (Doersch et al., 2015) are examples of auxiliary tasks for image classification. These auxiliary tasks are based on the assumption that solving the respective auxiliary task emphasizes the extraction of visual features, which can then be used to improve the performance for downstream tasks.

After a model has learned to solve an auxiliary task, its trained parameters can be used to initialize the parameters of a second model. This model is then trained on a downstream task using supervised learning. Compared to supervised learning from scratch, the model does not need to learn how to extract all required features from the limited labeled dataset. It can instead restrict itself to *finetune* the parameters for the given task. Finetuning can improve prediction performance with significantly fewer training samples. This makes it attractive for application scenarios with limited access to labeled data.

*Transfer learning* is an approach that is conceptually similar to SSL. Here, a model is first *pre-trained* on a different labeled dataset, which is typically larger than the available task-specific dataset. The *ImageNet* dataset (Russakovsky et al., 2015), consisting of over 14 million photos from 20 000 classes, is often used for pre-training. Most established deep learning software frameworks (Abadi et al., 2016; Paszke et al., 2019) make parameters of models trained on ImageNet easily available for many model architectures. Similar to SSL, transfer learning can improve prediction performance on downstream tasks and soften the requirements for large labeled datasets.

### 2.2.5 Visual representation learning

Learning good representations of images is of central importance for image analysis tasks (e.g., image classification, segmentation, or object detection). Visual representation learning aims to find a functional mapping from images to *feature vectors* in a low-dimensional *feature space*. Feature vectors encode relevant image features (e.g.,

textures or objects) in a compact form and provide the basis for different applications. For example, feature vectors may be used to classify images or find meaningful clusters in the data.

Visual representation learning is related to dimensionality reduction techniques, which also aim to identify relevant features and reduce the data dimensionality. For example, PCA (Pearson, 1901) linearly projects data points into a lower-dimensional feature space, making it an (admittedly simple) form of visual representation learning.

The challenge of visual representation learning lies in the definition of a learning objective (i.e., loss function) that promotes the extraction of useful features. Defining an adequate objective is particularly difficult when no labels are available to guide the learning process. Here, SSL methods can enable learning of meaningful features by defining auxiliary tasks that do not rely on labeled samples (Section 2.2.4).

The following sections introduce contrastive learning methods for visual representation learning. Contrastive visual representation has recently been shown to obtain good results for tasks like image classification and object detection (Chen et al., 2020). It provides the basis for the cytoarchitecture classification method presented in Chapter 6.

#### Contrastive representation learning

Hadsell et al. (2006) propose to use contrastive learning for visual representation learning. Unlike most other loss functions (e.g., categorical cross-entropy), which consider each data point in a dataset in isolation, contrastive learning uses a *contrastive loss* to compare (or *contrast*) multiple data points to each other. The contrastive loss measures the distance[22] between the features of different data points and is constructed such that the distances between "similar" data points are minimized, and the distances between "dissimilar" data points are maximized. Optimizing a differentiable function (e.g., a neural network) using the contrastive loss thus "encourages" it to map similar data points to nearby points in the feature space, while dissimilar points are mapped to different regions of the feature space. This disentanglement of data points in the feature space enables efficient classification or clustering. Hadsell et al. (2006) illustrate this idea by the analogy of a spring model, in which similar data points are pulled together, and dissimilar data points are pushed apart in the feature space (Figure 2.15). The definition of "similarity" and "distance" between data points depends on the given task and is a major aspect differentiating existing contrastive learning methods.

In recent years, several methods (van den Oord et al., 2019; Caron et al., 2020; Chen et al., 2020; He et al., 2020; Hénaff et al., 2020; Jain et al., 2020; Khosla et al., 2020) have adapted contrastive learning for visual representation learning to improve

[22]The term "distance" does not necessarily refer to the Euclidean distance.

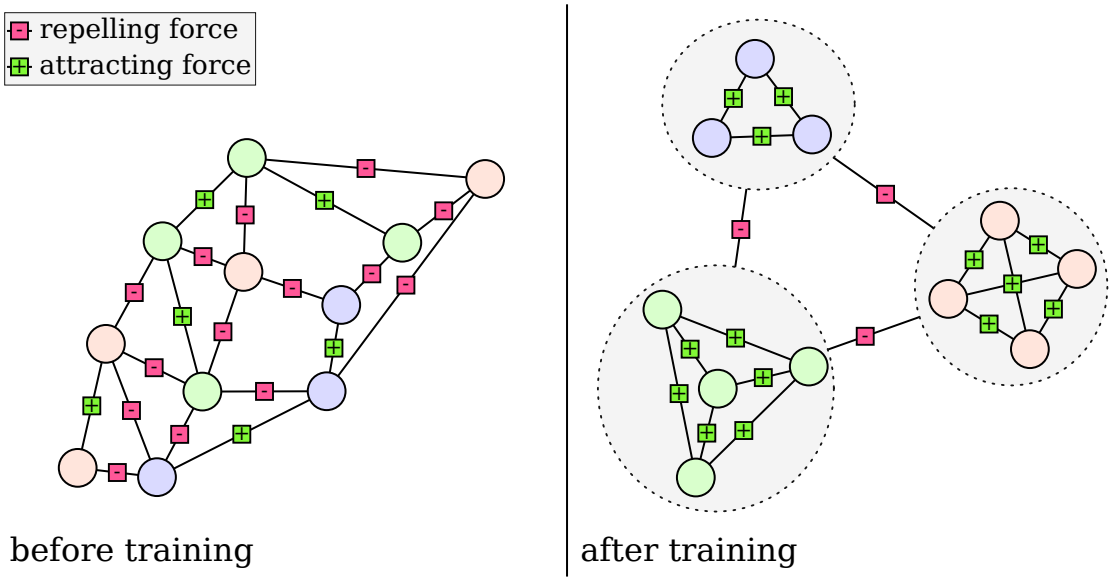



**Figure 2.15:** Illustration of contrastive representation learning. The circles represent data points. Colors indicate the similarity of the data points (i.e., points with the same color are considered similar). The definition of similarity depends on the specific method, e.g., based on data augmentation (Chen et al., 2020) or class labels (Khosla et al., 2020). **Left**: Before training, data points are scattered randomly in the feature space. The contrastive training optimizes the feature space (e.g., by optimizing the parameters of a deep neural network) to map similar data points to similar locations and dissimilar data points to dissimilar locations. Hadsell et al. (2006) illustrate this idea using the analogy of a spring system with repelling (red) and attracting (green) forces. Only a subset of possible connections between data points is drawn here to ensure clarity. **Right**: After training, the data points in the feature space are organized into clusters based on their similarity relationship. The learned feature space enables the separation of the data points (e.g., using a linear classifier) or data-driven exploration using clustering.

the performance of deep neural networks in different downstream tasks (e.g., classification or object detection). Most of these methods follow a common underlying approach, which consists of two stages (Figure 2.16): In the *pre-training* stage, a deep neural network is trained using contrastive learning to extract good visual representations. This stage aims to find a model parameterization that enables the mapping of images to compact feature vectors capturing relevant image features. In the following *finetuning* stage, the pre-trained model is used for a given downstream task (e.g., image classification). This can be achieved in multiple ways, e.g., by training a linear classifier on feature vectors (which is called *linear evaluation*), or by using the pre-trained parameters as initialization for a classification model (Section 2.2.4).

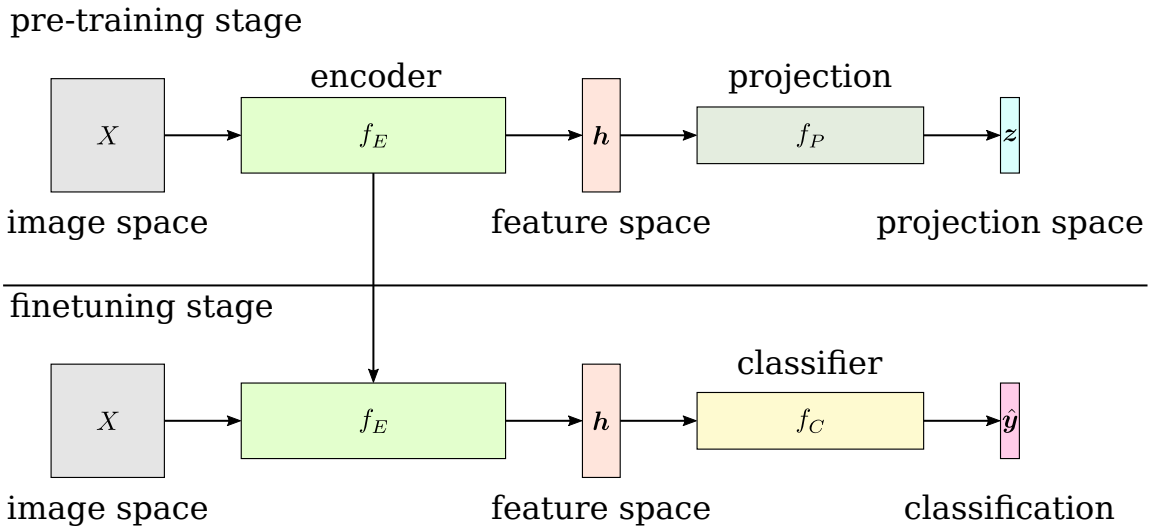



**Figure 2.16:** Illustration of the two-stage training workflow for contrastive learning. In the pre-training stage, an encoder model $f_E$ maps an image $X$ to a feature vector $\boldsymbol{h} \in \mathbb{R}^{d_h}$. A projector model $f_P$ then maps the feature vector to a projection space, where the resulting vectors $\boldsymbol{z} \in \mathbb{R}^{d_z}$ are used to compute and optimize a contrastive loss function. In the following finetuning stage, a linear classifier $f_C$ is trained based on features $\boldsymbol{h}$ extracted by the encoder $f_E$.

This two-stage approach decouples feature learning from the downstream task and has shown to achieve good performance for different applications (Chen et al., 2020).

#### Self-supervised contrastive learning

Parts of the success of contrastive learning methods can be attributed to the upcoming of self supervised contrastive learning methods. These methods perform contrastive visual representation learning without relying on annotated image data (i.e., class labels), making them applicable to very large image datasets.

The SimCLR method (Chen et al., 2020) is an important self supervised contrastive learning method. SimCLR exploits data augmentation to define the similarity relationship between images: In a first step, two *views* $\tilde{X}_i$ and $\tilde{X}_{j(i)}$ are created by applying two different data augmentation operations to an image $X_i$ . $j(i)$ specifies the index of the respective other view in a batch of training samples. The assumption is that two views created from the same source image can be considered similar because the data augmentation operation varies irrelevant image features and keeps relevant features intact. SimCLR optimizes a contrastive loss function to map views of the same source image (i.e., "similar" samples) to similar regions of the feature space and views from different source images to different regions of the feature space.

Chen et al. (2020) show that their approach achieves superior linear evaluation classification performance compared to other SSL approaches or learning from scratch.

SimCLR and other contrastive learning methods (Wu et al., 2018; van den Oord et al., 2019; He et al., 2020; Hénaff et al., 2020) use the normalized temperature-scaled cross-entropy (NT-Xent) loss, also referred to as *InfoNCE* loss. In the case of SimCLR, the NT-Xent loss is computed by sampling a batch of $b$ images

$$\mathbb{B} = \{X_i, \cdots, X_b\}, \tag{2.51}$$

from which a batch

$$\tilde{\mathbb{B}} = \left\{\tilde{X}_1, \cdots, \tilde{X}_b, \tilde{X}_{j(1)}, \cdots, \tilde{X}_{j(b)}\right\} \tag{2.52}$$

containing $2b$ views is derived. The NT-Xent loss is then defined as

$$L^{\text{ssl}}\left(\tilde{\mathbb{B}}\right) = \sum_{i=1}^{2b} l^{\text{ssl}}(i) \tag{2.53}$$

$$\text{with } l^{\text{ssl}}(i) = -\log \frac{\exp\left(\text{sim}\left(\boldsymbol{z}_i, \boldsymbol{z}_{j(i)}\right)/\tau\right)}{\sum_{k=1}^{2b} \mathbb{I}_{i \neq k} \exp\left(\text{sim}\left(\boldsymbol{z}_i, \boldsymbol{z}_k\right)/\tau\right)}, \tag{2.54}$$

where $\boldsymbol{z}_i \in \mathbb{R}^{d_z}$ $(1 \leq i \leq 2b)$ are computed as

$$\boldsymbol{z}_i = f_P\left(f_E\left(\tilde{X}_i\right)\right). \tag{2.55}$$

Both $f_E$ and $f_P$ are deep neural networks. The *encoder*

$$f_E\left(\tilde{X}_i\right) = \boldsymbol{h}_i \in \mathbb{R}^{d_h} \tag{2.56}$$

maps an image $\tilde{X}_i$ to a feature vector $\boldsymbol{h}_i \in \mathbb{R}^{d_h}$. The *projection head*

$$f_P(\boldsymbol{h}_i) = \boldsymbol{z}_i \in \mathbb{R}^{d_z} \tag{2.57}$$

then maps the feature vector $\boldsymbol{h}_i$ to a projection space where the contrastive loss is computed. Chen et al. (2020) found that using the projection head to project feature vectors to a lower-dimensional space before computing the contrastive loss leads to better performance compared to computing the contrastive loss directly on the feature vectors. $\text{sim}(\boldsymbol{u}, \boldsymbol{v})$ is a function that measures the similarity or "distance" between two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$. A typical example is the *cosine similarity*

$$\text{sim}(\boldsymbol{u}, \boldsymbol{v}) = \frac{\langle \boldsymbol{u}, \boldsymbol{v} \rangle}{\|\boldsymbol{u}\|_2 \|\boldsymbol{v}\|_2}, \tag{2.58}$$

which measures the cosine of the angle between two vectors. The *temperature* $\tau \in [0, 1]$ controls the balance between positive and negative pairs: Higher values emphasize the similarity of positive pairs, while lower values emphasize the dissimilarity of negative pairs.

The vectors $\boldsymbol{z}_i$ and $\boldsymbol{z}_{j(i)}$ in the numerator represent a *positive pair* of data points, i.e., data points that should be mapped to similar feature representations. The denominator sums up the similarity between $\boldsymbol{z}_i$ and all other samples, referred to as *negative pairs*. Maximizing the numerator and minimizing the denominator consequently encourages the deep neural networks $f_E$ and $f_P$ to produce similar feature vectors for similar samples and dissimilar feature vectors for dissimilar samples.

SimCLR uses samples *within a batch* to create negative pairs. As a result, SimCLR relies on a sufficiently large batch size to provide enough negative examples. Using large batch sizes can be technically challenging, in particular when the size of each sample (i.e., image) is also large.

After the pre-training stage with contrastive learning, SimCLR discards the projection head $f_P$ and uses the encoder $f_E$ to produce features for the following finetuning stage. Chen et al. (2020) use the *linear evaluation* protocol to evaluate the quality of learned features. Here, a linear classifier $f_C$ is trained to perform a downstream task (e.g., classification) based on the learned features. Figure 2.16 illustrates this two-stage training workflow.

The idea behind SimCLR is representative for many other contrastive learning approaches. Other approaches mainly differ in the way they define similarity (Hénaff et al., 2020) or how negative pairs are created (van den Oord et al., 2019; He et al., 2020). For example, the *momentum contrast* approach proposed in He et al. (2020) maintains a queue of past training batches, which are used to provide negative pairs for the contrastive loss. This approach addresses one of the shortcomings of SimCLR, which depends on the use of large batch sizes to provide a sufficient amount of negative pairs for training. Another example is given by van den Oord et al. (2019), who consider two non-overlapping patches as similar if they were extracted from the same image. Like SimCLR, they create negative examples from other images in the same batch.

### Supervised contrastive learning

Many contrastive learning methods (van den Oord et al., 2019; Chen et al., 2020; He et al., 2020; Chen et al., 2021) do not rely on class labels and can thus be regarded as SSL approaches. For example, SimCLR (Chen et al., 2020) defines similarity solely based on data augmentation, so the contrastive pre-training does not rely on class labels.

Khosla et al. (2020) point out that the data augmentation-based similarity relationship proposed by Chen et al. (2020) has a high risk of producing false-negative pairs. SimCLR does not take class labels into account, so different images belonging to the same class (e.g., containing the same object) are considered dissimilar. A model trained using the SimCLR framework might map images containing the same class to feature vectors in distant regions of the feature space, which can negatively affect the downstream task performance.

Khosla et al. (2020) propose *supervised contrastive learning*, which combines a contrastive loss with a supervised training strategy. Using the notation from Equation 2.54, the *supervised contrastive loss* is defined as

$$L^{\text{ssl+}}\left(\tilde{\mathbb{B}}\right) = \sum_{i=1}^{2b} l^{\text{ssl+}}(i)\,, \tag{2.59}$$

$$\text{with } l^{\text{ssl+}}(i) = -\frac{1}{2n_{c_i} - 1} \sum_{j=1}^{2b} \mathbb{I}_{i \neq j} \mathbb{I}_{c_i = c_j} \log \frac{\exp\big(\text{sim}\big(\boldsymbol{z}_i, \boldsymbol{z}_j\big)/\tau\big)}{\sum_{k=1}^{2b} \mathbb{I}_{i \neq k} \exp\big(\text{sim}\big(\boldsymbol{z}_i, \boldsymbol{z}_k\big)/\tau\big)}, \tag{2.60}$$

where $c_i$ is the class of the image with index $i$ and $n_{c_i}$ is the number of samples in the batch belonging to $c_i$. In this supervised variant of the contrastive loss used by SimCLR (Chen et al., 2020), two views are similar if they are created from the same source image (just like in SimCLR) *or* if they belong to the same class. This approach addresses the issue of false-negative pairs that can occur in SSL approaches, at the cost of requiring class labels. Note that the definition does not need to explicitly model the similarity relationship of views, as views created from the same source image share the same label and are thus considered similar based on the class labels.

Khosla et al. (2020) show that training with Equation 2.59 leads to significantly improved performance over training with categorical cross-entropy. They also demonstrate that training with supervised contrastive loss is less sensitive to changes in the hyperparameter configuration (e.g., the learning rate, the optimizer, the kind of data augmentation), making it easier to find a suitable hyperparameter configuration for a given task. Finally, they proof that Equation 2.59 is a generalization of the *triplet loss* (Weinberger et al., 2009), which is an alternative to categorical cross-entropy for learning robust representations in supervised settings (Khosla et al., 2020). Along these lines, Marrakchi et al. (2021) showed that supervised contrastive learning helps to mitigate class imbalance and achieve improved performance compared to training with categorical cross-entropy.

### 2.2.6 Technical aspects

The growing availability of powerful compute resources played an important role in the recent advent of deep learning (Krizhevsky et al., 2012; Goodfellow et al., 2014). In particular, the availability of graphics processing units (GPUs) has gained significant importance for the development and application of deep learning algorithms. GPUs were initially developed for computer graphics applications (e.g., 3D rendering or video games). Their ability to apply a computing operation to different data points in parallel (*single instruction, multiple data (SIMD)* architecture) facilitates high-performance rasterization of 3D scenes. The same property can be exploited to perform many operations used in deep neural networks (e.g., matrix multiplications or convolutions) in an optimized way. Their availability as a standardized and relatively cheap consumer product led to the widespread adoption of GPUs for deep learning applications in research and industry. Software frameworks for deep learning (Abadi et al., 2016; Al-Rfou et al., 2016; Paszke et al., 2019) support the use of GPUs, mostly without requiring the user to deal with GPU programming. Modern GPUs even provide dedicated hardware units for deep learning, e.g., the *tensor cores* of NVidia GPUs.

The training phase is the most time-intensive part of developing a deep learning model. Suitable model architectures and hyperparameters for a given task must typically be identified using trial-and-error. The iterative development workflow relies on the ability to train models in reasonable time frames.

Training a model on a large dataset can require several days or weeks of computation when using a single GPU, which significantly slows down the development process. Distributed deep learning (DDL) enables the training of deep neural networks using multiple GPUs that are installed in the same or different interconnected compute nodes. It allows combining the computational power and the memory of multiple GPUs, which is crucial to use large model architectures, large batch sizes, large image sizes, and to reduce the total training time. DDL can be implemented using different parallelization paradigms, the two most important being *model parallelism* and *data parallelism* (Figure 2.17). Data parallel training works by subdividing a batch of samples that might not fit into the memory of a single GPU into equally sized sub-batches. The sub-batches can be processed independently by multiple GPUs. In the backpropagation step, the gradients computed by each GPU (Equation 2.5) are averaged before updating the model parameters (Equation 2.3). This approach virtually combines the computational power and memory of the available GPUs. The gradient averaging step incurs a communication overhead, so performance does not necessarily scale linearly with the number of used GPUs.

Model parallel training instead distributes the layers of a deep neural network across GPUs. For example, a model can be distributed by storing and computing

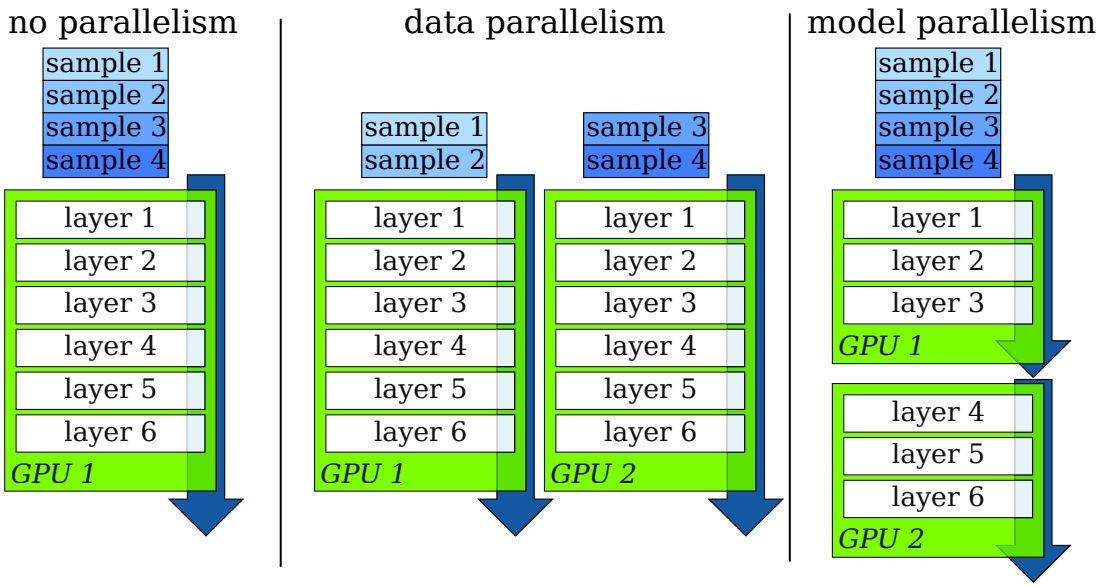



**Figure 2.17:** Schematic illustration of different DDL approaches. Without DDL (left), all samples of a batch are processed by all layers of a model on one GPU. Using data parallelism (middle), each GPU processes a subset of samples in a batch in parallel. Gradients computed per GPU are averaged across GPUs during the backpropagation step. Using model parallelism (right), each GPU computes the output of a subset of layers, requiring inter-GPU communication at points where the model was split.

half of the model's layers on one GPU each. The first GPU takes the input and passes it through the first half of layers. The obtained intermediate results are transferred to the second GPU, which uses them to compute the final output. Model parallel training combines the memory of multiple GPUs. However, the temporal dependence of the computations (i.e., earlier layers have to be computed before later layers) prevents a compute time reduction when using naive model parallelism[23]. Pipelining methods (Huang et al., 2019) can mitigate this effect through efficient scheduling of computations.

Large models that occupy more than the accumulated memory capacity of all available GPUs can be trained using gradient checkpointing (Chen et al., 2016b). Gradient checkpointing is a strategy that reduces the memory requirements for neural network training at the cost of increased runtime. Gradient computation using the backpropagation algorithm relies on intermediate results from the forward pass. By default, these results are kept in memory after they are computed. Gradient checkpointing designates specific intermediate results as *checkpoints* and discards all but

[23] The *AlexNet* (Krizhevsky et al., 2012) is an example for a model architecture that is explicitly designed to benefit from model parallelism. The model consists of two branches of layers, which can be computed largely independent on separate GPUs.

these results after the forward pass. During the backward pass, discarded results are recomputed from the stored checkpoint results. As a result, gradient checkpointing enables the training of large models that would otherwise not fit into memory.

### 2.2.7 Deep learning for cytoarchitecture analysis

Deep learning has already been successfully applied for different cytoarchitecture analysis tasks.

Atzeni et al. (2018) proposed a probabilistic framework to combine CNNs with an atlas registration approach for segmenting different tissue types in histological brain sections. They demonstrate that their method can reliably identify white matter, ventricles, the brainstem, subcortical and cortical gray matter in images of the AAHA. However, they note that their method is not able to distinguish more subtle subdivisions of those classes, in particular detailed cytoarchitectonic areas (Atzeni et al., 2018). In addition, the high computational requirements of the method force it to operate at a resolution of 250 µm/px, which is not sufficient to identify cytoarchitectonic areas.

Wagstyl et al. (2020) present a method for automatic segmentation of cortical layers in the 3D BigBrain model using a one-dimensional CNN. They propose to compute a set of traverses through the cortex, each running perpendicular to both pial boundary and gray-white matter boundary. Voxel intensities are then sampled along these traverses, resulting in profiles characterizing the laminar composition at different positions in the cortex. Using a set of labels created from manual annotations, they then train a one-dimensional CNN to segment cortical layers in the BigBrain model. The resulting cortical layer atlas facilitates structural analysis of layers in the BigBrain model (Wagstyl et al., 2020).

The work by Wagstyl et al. (2020) is an important example of how deep learning can help to analyze cytoarchitecture at a large scale. Their method exploits the availability of the precise 3D reconstruction provided by the BigBrain model. However, the development of analysis methods that can be applied to incomplete series of 2D microscopic scans remains a challenge. Furthermore, the classification of cytoarchitectonic areas relies on the ability to identify individual cells and thus requires image data at an even higher resolution than currently provided by the BigBrain model.

Spitzer et al. (2017) were the first to investigate deep learning for automatic classification of cytoarchitectonic areas. They formulate cytoarchitecture classification as a segmentation problem, with the goal to assign each pixel in an input image to the corresponding cytoarchitectonic area. To handle the large amounts of high-resolution image data that need to be analyzed to address this task, their processing pipeline operates on rectangular patches extracted from whole-slide images. Image patches are extracted at a resolution of 2 µm/px and with a size of approximately $2000 \times 2000$

pixels. The large size and high resolution of each image patch ensure that details of the cellular composition across the whole extent of the cortex can be captured and used for classification. Spitzer et al. (2017) propose a modified U-Net architecture that is capable of handling such large image patches. For their experiments, they use a dataset of 111 cell-body stained histological human brain sections with annotations of 13 cytoarchitectonic areas from the visual system of the human brain.

Their work revealed how challenging cytoarchitectonic mapping is as an image segmentation problem. Spitzer et al. (2017) show that the inclusion of additional input information extracted from the probabilistic Julich-Brain atlas can considerably improve the prediction performance, as it helps to disambiguate complex patterns in the image data.

Building upon this work, Spitzer et al. (2018b) propose a SSL task to learn meaningful visual representations and thus improve the prediction performance. They define an auxiliary task by exploiting the 3D reconstruction of the BigBrain dataset: Given two images patches extracted from two positions within the cortex of the BigBrain model, a CNN is tasked to predict the geodesic distance between the two patches along the brain surface, as well as the position of the respective patches in the BigBrain reference space. This auxiliary task exploits that close locations in the cortex are more likely to come from the same or similar cytoarchitectonic areas than spatially distant locations. Thus, it can be assumed that a model trained to predict patch location and distances between patch pairs learns features that are also relevant for the downstream task of classifying cytoarchitectonic areas. Results presented by Spitzer et al. (2018b) show that models that were pre-trained in this way achieve superior performance compared to models that were trained from scratch. These results indicate that the proposed SSL task enables learning of meaningful cytoarchitectonic features. In a follow-up work (Spitzer et al., 2018a), the authors show that learned features even tend to form anatomically meaningful clusters (e.g., corresponding to different lobes).

The methods presented in this thesis build upon these foundations to classify more cytoarchitectonic areas, improve the classification accuracy, and develop practical methods for large-scale cytoarchitectonic mapping.

# 3 Microscopic image datasets

This chapter introduces the data used in this thesis. Section 3.1 details the processing protocol for acquiring the microscopic images. Section 3.2 introduces used datasets. Section 3.3 gives an overview of annotated cytoarchitectonic areas.

## 3.1 Histological processing of human brain sections

Analysis of high-resolution microscopic scans of cell body stained histological whole-brain sections is considered the gold standard for identifying cytoarchitectonic areas. This method provides high resolution and high contrast for neuronal cell bodies, enabling the study of subtle variations in the cortical cytoarchitecture. The steps required to prepare microscopic scans are detailed in the following.

This work uses microscopic images acquired from postmortem human brains. The brains are part of the brain collections of the Institute of Neuroscience and Medicine (INM-1) at Forschungszentrum Jülich (FZJ) (Germany) and the Cécile and Oskar Vogt Institute for Brain Research at University Hospital of Heinrich-Heine University Düsseldorf (Germany). The brains were obtained through the body donor programs of the anatomical institutes of the universities of Düsseldorf, Rostock, and Aachen. All brain samples were acquired in accordance with legal and ethical regulations and guidelines. The studies carried out require no ethical approval. All body donors have signed a declaration of agreement.

The autopsies were performed within 24 hours after death[1]. After extraction from the skull, the brains were chemically fixated (e.g., using formalin) to limit shape distortions. Brains were then embedded in paraffin to prepare them for the sectioning process. Each brain was sectioned into 6000-7500 coronal histological sections with a thickness of 20 µm each. Every 15th section was mounted on a glass slide and stained for cell bodies. For one brain (B20), all obtained sections were mounted and stained. A modified silver staining (Merker, 1983) was used to stain neuronal cell bodies, enabling the analysis of the neuronal cell distribution.

Prepared sections were digitized using high-throughput light-microscopic scanners (TissueScope HS, Huron Digital Pathology Inc.). The resulting microscopic scans have a resolution of 1 µm/px, allowing analysis of the shape of individual cells and

[1] A detailed description of the processing protocol is given in Amunts et al. (2000).

**Table 3.1:** List of brains used in experiments. Each brain is identified by an integer. The table lists the age of death and gender of the brain donors and the fresh weight of each brain after autopsy. The shrinkage factor specifies the ratio between the fresh brain volume and the volume after histological processing. The number of sections specifies how many sections are available as microscopic images.

| Brain | Age (years) | Gender | Fresh weight (g) | Shrinkage | # Sections |
|---|---|---|---|---|---|
| B01 | 79 | F | 1350 | 1.933 | 471 |
| B03 | 69 | M | 1360 | 2.129 | 509 |
| B04 | 75 | M | 1349 | 2.153 | 497 |
| B05 | 59 | F | 1142 | 2.056 | 454 |
| B06 | 54 | M | 1757 | 2.446 | 479 |
| B07 | 37 | M | 1437 | 2.024 | 489 |
| B10 | 85 | F | 1046 | 1.671 | 493 |
| B12 | 43 | F | 1198 | 2.062 | 467 |
| B20 | 65 | M | 1392 | 1.931 | 7404 |

their distribution. Images have a median size of 77 000 px × 105 000 px, with a maximum size of up to 95 000 px × 136 000 px.

## 3.2 Brain samples & datasets

The experiments conducted in this work use data from nine postmortem adult human brains (Figure 3.1) provided by the INM-1, which were acquired using the protocol described in Section 3.1. Each brain is identified by an integer, which is denoted as B (e.g., B01 for the brain with identifier 1). Table 3.1 lists the age of death and the gender of the donor, the fresh weight of the brain after the autopsy, the volumetric brain shrinkage factor, and the number of available microscopic images for each brain. The shrinkage factor is defined as the ratio between the volume of a brain before and after histological processing. Histological processing leads to shrinkage of the brain tissue. The tissue shrinkage needs to be considered when computing geometric measures (e.g., distances, areas, or volumes) in processed postmortem brains. Approximately every 15th brain section was digitized. For B20, all obtained sections were digitized. B20 comprises the sections that were used to create the BigBrain model (Amunts et al., 2013).

We denote a microscopic scan of a histological brain section as image

$$S_s^{\mathtt{B}} \in [0, 255]^{h_s^{\mathtt{B}} \times w_s^{\mathtt{B}}}. \tag{3.61}$$

Here, B denotes the brain sample the section belongs to (e.g., B01). The *section number* $s \in \mathbb{N}$ identifies a specific microscopic image in the section stack. Section

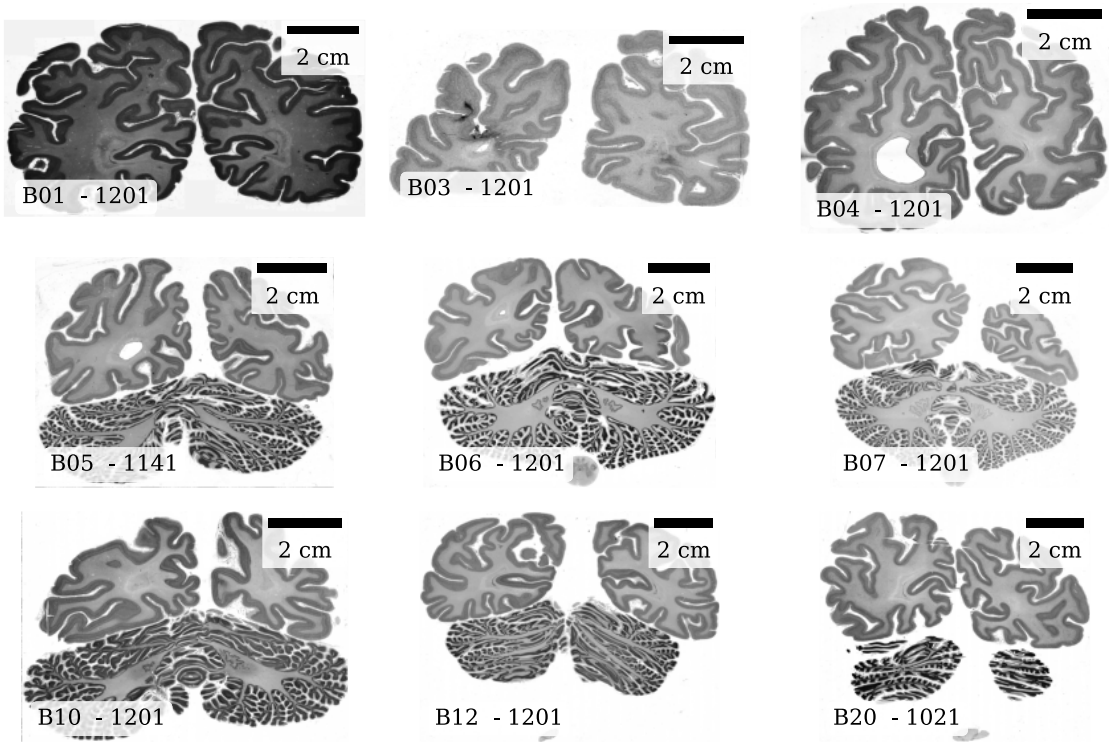



**Figure 3.1:** Microscopic images of coronal histological brain sections from comparable locations in the occipital lobe of nine postmortem human brains. The examples illustrate the variability between different brains with respect to staining and tissue morphology. Brains B05 to B20 were imaged including the cerebellum, which is visible in the lower half of each image.

numbers increase from the posterior pole to the anterior pole. The height (number of rows) and width (number of columns) of an image are denoted as $h_s^{\mathtt{B}}$ and $w_s^{\mathtt{B}}$, respectively.

### Tissue segmentation

The tissue of the cerebrum can be subdivided into gray matter and white matter tissue (Section 2.1). We segment the microscopic images (Figure 3.2, A) into gray matter, white matter, and *background*. The latter comprises the surrounding microscopy slide on which the tissue is mounted. Tissue segmentations allow targeted processing of specific tissue classes, for example exclusion of background or analysis of gray matter tissue. Compared to cytoarchitectonic areas, which are defined by subtle variations of cytoarchitecture, tissue classes show distinct intensity distributions and are relatively easy to segment.

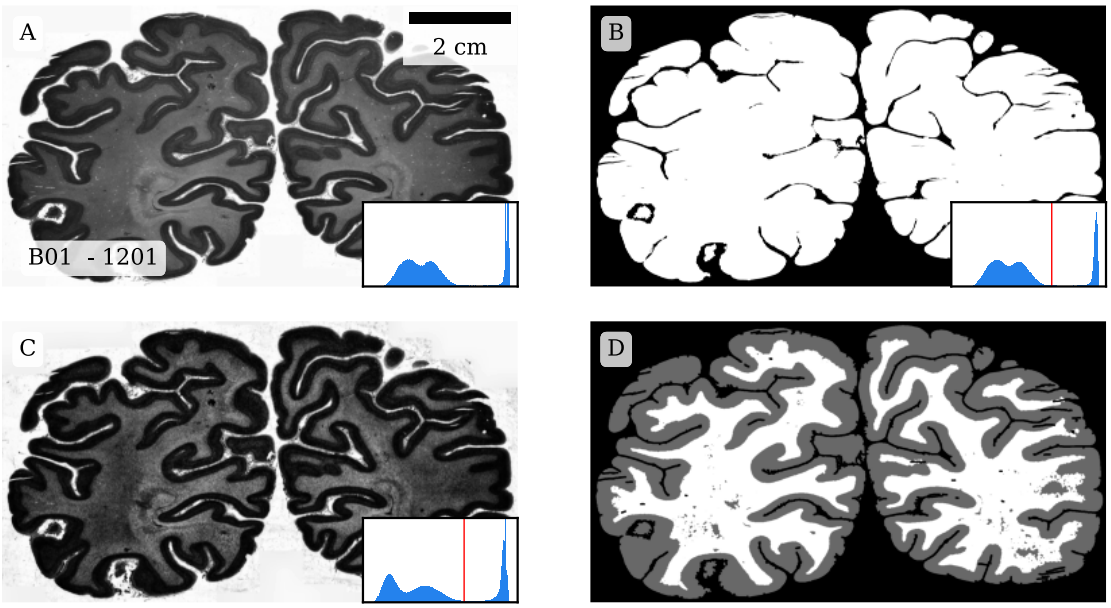



**Figure 3.2:** Intermediate results of the tissue segmentation procedure for section 1201 of B01. **A**: Original microscopic image of a histological brain section. **B**: Segmentation into tissue (white) and background (black). **C**: Contrast enhanced input image with improved separability of cortex and white matter. **D**: Final segmentation into background (black), white matter (white) and gray matter (gray). Cortex and white matter are separated by morphological active contours (Márquez-Neila et al., 2014). Intensity histograms show the respective pixel intensity distribution of the original image (A), after smoothing for background-tissue segmentation (B), and after contrast enhancement for gray-white matter segmentation (C). Red lines mark the threshold between background and tissue.

Tissue segmentation is performed on downscaled images (64 µm/px). Pixels are first segmented into background and tissue pixels (Figure 3.2, B). Background and tissue show distinct intensity distributions and are thus approximately separable by intensity thresholding. The threshold is determined per section by searching for local minima in the intensity histogram of the image (256 bins). The intensity histogram is smoothed using a median filter (size 3) and a mean filter (size 5) to make the process robust against noise. If more than one minimum is found, the one closest to the Otsu threshold (Otsu, 1979) is used.

Tissue pixels are then separated into gray matter and white matter. The separation is performed by morphological active contours (Márquez-Neila et al., 2014), a variant of the Chan-Vese segmentation method (Chan et al., 1999), which is is suitable for segmenting images without well-defined borders. Before segmentation, the image contrast is enhanced to emphasize the intensity difference between gray matter and

white matter. We first apply a minimum filter, a maximum filter, and a mean filter (each with size 5). The contrast is then enhanced using contrast limited adaptive histogram equalization (CLAHE, Pizer et al., 1987) with kernel size 250, followed by Gaussian blurring (standard deviation 1). The result is again filtered using a minimum, a maximum, and a mean filter (size 5). The obtained segmentation masks are cleaned using morphological operations to remove small objects and holes from the masks (Figure 3.2, D). Particular care is taken to prevent tight sulci from being closed during segmentation or the subsequent cleaning. This step is important to retain the shape of the cortical ribbon.

The described tissue segmentation procedure is applied for all brains but B20. Here, Lewis et al. (2014) provide a publicly available tissue segmentation for the 3D BigBrain model, which we transform onto the microscopic images of B20 (Section 3.2). The quality of the resulting tissue segmentations is comparable to that obtained using the above procedure.

We visually inspect the segmentation results for all considered sections to evaluate their quality. The segmentations are sufficient to approximately locate the cortex in the microscopic images (Section 5.1.2) and recover its three-dimensional structure (Section 7.1.1). More elaborate approaches (e.g., based on machine learning) could potentially improve the segmentation quality, but the described procedure is sufficient for the considered use cases.

### Rigid alignment of brain sections

Cutting a brain into thin histological sections inevitably introduces physical deformations of the tissue. Accurate reconstruction of its original 3D shape from individual serial sections relies on complex, nonlinear reconstruction strategies (Amunts et al., 2013). Fully automated methods for precise 3D brain reconstruction that can be reliably applied to whole human brain sections are not available. We thus rely on an approximate *linear* alignment of brain sections, which is easier to obtain and provides sufficient precision for the use cases presented in this work (Sections 5.1.2 and 7.1.1).

We compute linear transformations to approximately align consecutive brain sections. A linear transformation

$$\Phi(\boldsymbol{x}; M) = M\boldsymbol{x} \tag{3.62}$$

transforms a homogeneous coordinate $\boldsymbol{x} \in \mathbb{R}^3$ by multiplication with a linear *transformation matrix* $M \in \mathbb{R}^{3\times3}$. *Registration* methods estimate the parameters of $M$ such that $\Phi$ maps points from one image[2] to approximately corresponding points in

[2] We consider the 2D case (i.e., images), but the approach is also applicable to higher-dimensional data (e.g., 3D volumes).

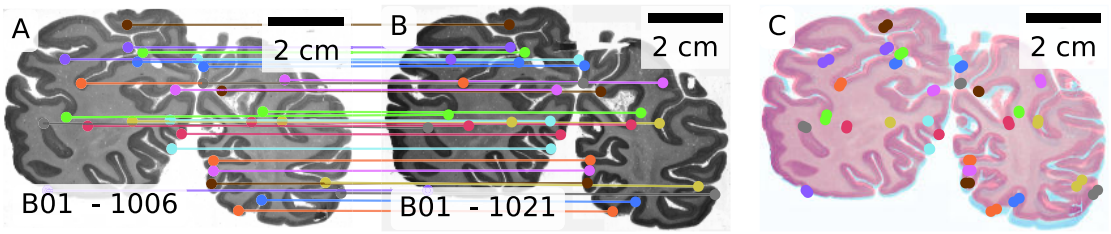



**Figure 3.3:** Rigid alignment of the consecutive histological sections 1006 and 1021 from B01. Sections are linearly aligned by feature-based image registration as described in Dickscheid et al. (2019). **A+B**: Prominent features (colored dots) are detected using a SURF feature detector (Bay et al., 2006). Features are matched using k-nearest neighbor matching and the RANSAC algorithm (Fischler et al., 1981). Matched points are indicated with colored lines connecting features detected in both sections. These matches are used to compute a rigid transformation that aligns the two sections. Only 30 of the found matches are shown in this example for better visibility. **C**: Sections that were aligned relative to section 1021 using the computed rigid transformation. Section 1006 and 1021 are encoded in the red and blue channels of the image, respectively.

a second image. Depending on the use case, the transformation can be restricted to specific geometric transformations. For example, *rigid* (or *Euclidean*) transformations include rotation and translation, while *affine* transformations additionally include reflection (mirroring), scaling, and shearing. Rigid transformations preserve angles and distances. Affine transformations preserve lines and length ratios.

Linear transformations are generally not sufficient to accurately reconstruct a brain's 3D shape from individual sections. However, a linear alignment is considerably easier to achieve than a precise nonlinear reconstruction. The provided precision is sufficient for our use cases, which include identification of approximately corresponding regions (Section 5.1.2) and creation of coarse 3D brain reconstructions (Section 7.1.1).

We estimate the parameters of a linear transformation $\Phi_{s \to t}$ between each pair of consecutive brain sections $s$ and $t$, such that each point in $s$ is mapped to the approximate corresponding point in $t$. Dickscheid et al. (2019) describe a feature-based image registration method for approximate alignment of histological brains sections, which we adapt to compute a rigid alignment between consecutive brain sections.

Images are downscaled to 64 µm/px resolution and smoothed using a median filter. A SURF feature detector (Bay et al., 2006) is applied to detect prominent features in both images. Potentially corresponding features are detected using k-nearest neighbor matching. The RANSAC algorithm (Fischler et al., 1981) then determines a

robust set of matches. Finally, a rigid transformation is estimated from the detected matches using least-squares fitting.

We assess the alignment quality by inspecting overlays of aligned consecutive sections (Figure 3.3, C). In most cases, rigid transformations are sufficient to approximately align consecutive sections. A few brain sections were imaged from the "wrong" side, resulting in mirrored images. We identify these images during the quality control and resolve the issue by aligning them using affine transformations.

Note that transformations between all pairs of *adjacent* sections enable transformations between *arbitrary* pairs of sections, as the expression $\Phi_{s\to s''} = \Phi_{s\to s'} \circ \Phi_{s'\to s''}$ can be recursively applied for arbitrary $s < s' < s''$ and $\Phi_{t\to s} = \Phi_{s\to t}^{-1}$ holds for arbitrary choices of $s$ and $t$.

### Data from the Julich-Brain atlas

We use the Julich-Brain probabilistic atlas (Amunts et al., 2020) to project probabilistic cytoarchitectonic maps (Figures 2.2 and 3.4) and canonical spatial coordinates onto the histological brain sections. The projected data provides input features for the cytoarchitecture classification method proposed in Chapter 7.

The Julich-Brain atlas was created from annotated cytoarchitectonic areas in 23 human brains (Section 2.1.3). The individual sections were reconstructed[3] to create a 3D volume for each brain. The reconstruction comprises linear and nonlinear alignment of consecutive sections, followed by a volume-based elastic 3D registration (Hömke, 2006) to MRI volumes that were acquired prior to cutting. The reconstruction workflow is used to create three-dimensional maps of cytoarchitectonic areas for each brain, which are then projected (Operto et al., 2008) into the MNI-Colin27 (Holmes et al., 1998) and the *ICBM152casym* (Fonov et al., 2011) reference spaces (Section 2.1.3). The transformed maps are superimposed to create probabilistic maps (Figures 2.2 and 3.4).

The brain samples considered in this work (Section 3.2) represent a subset of the brains that were used to create Julich-Brain. Thus, the above reconstruction workflow allows transforming data from the three-dimensional reference spaces onto individual histological sections and vice versa. We exploit this relationship to project data defined in the MNI-Colin27 space onto the histological sections. The MNI-Colin27 reference space is created from MRI measurements of a single subject.

**Cytoarchitectonic probabilistic maps** We project the probabilistic maps of the Julich-Brain atlas (Figure 2.2) from the MNI-Colin27 reference space onto the histological brain sections. This step allows associating each pixel in an image with a prior probability distribution over the cytoarchitectonic areas that might occur at

[3] A detailed description of the reconstruction workflow is given in Amunts et al. (2020).

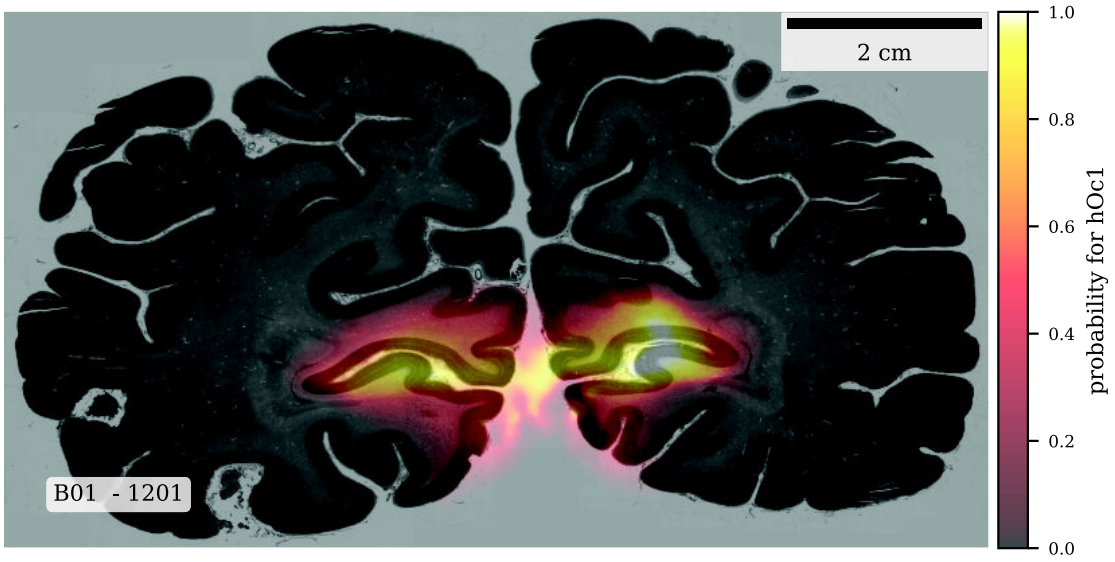



**Figure 3.4:** Probabilistic map of area `hOc1` projected from the Julich-Brain probabilistic atlas (Amunts et al., 2020) onto the histological section 1201 of `B01`. Bright values indicate locations where the occurrence of `hOc1` is likely. The spatial precision is limited but can serve as a prior for cytoarchitecture classification (Spitzer et al., 2017). Figure 2.2 (left) shows the corresponding three-dimensional probabilistic map in the MNI-Colin27 space.

the respective location (Figure 3.4). Spitzer et al. (2017) describe how prior information from probabilistic maps can be exploited to provide CNNs for cytoarchitecture classification with additional information. We adapt this idea and project probabilistic maps of 152 cytoarchitectonic areas from Julich-Brain onto the histological brain sections. The projected probabilistic maps for a section $s$ from brain $\mathtt{B}$ are represented as an image

$$P_s^{\mathtt{B}} \in [0,1]^{h_s^{\mathtt{B}} \times w_s^{\mathtt{B}} \times 152} \tag{3.63}$$

with 152 channels, where each channel represents the probabilistic map of a specific cytoarchitectonic area.

**Canonical spatial coordinates** Information on the spatial location of an image patch in the brain can help to disambiguate cytoarchitectonic patterns. For example, an image patch from the anterior part of the brain will never show visual areas, as these are located in the posterior part of the brain. Using spatial locations in algorithmic approaches relies on a well-defined coordinate system that is consistent across all considered brains. We use the correspondence between the histological

sections and the MNI-Colin27 space to define such a consistent *canonical coordinate system* across brains.

We define a coordinate system by assigning a 3D coordinate to each voxel of the MNI-Colin27 space. The coordinate system follows the RAS coordinate convention (i.e., the three axes correspond to the left to right, posterior to anterior, and inferior to superior directions, respectively). Coordinates are normalized to the range $[-1, +1]$. We then project the defined coordinates onto the histological brain sections of each considered brain. The projected canonical coordinates for a section $s$ from brain $\texttt{B}$ are represented as an image

$$R_s^{\texttt{B}} \in [-1, +1]^{h_s^{\texttt{B}} \times w_s^{\texttt{B}} \times 3} \tag{3.64}$$

with 3 channels, where each channel represents a spatial dimension in RAS coordinate convention. The projected coordinates enable identification of anatomically corresponding points in different brains (e.g., $R_s^{\texttt{B}}(i, j) \approx R_{s'}^{\texttt{B}'}(i', j')$), even in cases where the brain-specific location parameters (e.g., $s, i, j$ and $s', i', j'$) differ.

#### 3D reconstruction of the BigBrain

BigBrain (Amunts et al., 2020) is a high-resolution human brain atlas created by reconstructing 7404 histological sections into a consistent 3D brain model (Section 2.1.3). The histological sections from which the BigBrain model was originally created were rescanned at 1 µm/px resolution. These sections form the B20 dataset used in this work. The known correspondence between the B20 dataset and the 3D BigBrain model enables the transformation of data from the 3D BigBrain space onto the histological sections of B20, and vice-versa.

## 3.3 Annotations of cytoarchitectonic brain areas

Annotations of cytoarchitectonic areas are required for supervising the training of deep neural networks for automatic cytoarchitecture classification (Spitzer et al., 2017, 2018b). Borders between adjacent cytoarchitectonic areas are identified by the method described in Section 2.1.4 (Schleicher et al., 1999). The annotations used in this work were acquired over multiple decades in the scope of the JulichBrain (Amunts et al., 2020) project.

In practice, areas are often annotated in the scope of research projects that consider one or a few areas. Cytoarchitectonic areas are typically annotated in every 15-60th section. However, the subset of considered sections varies between areas. Areas are typically annotated in sections where cytoarchitecture can be reliably identified, which can vary depending on an area's location, size, the presence of histological

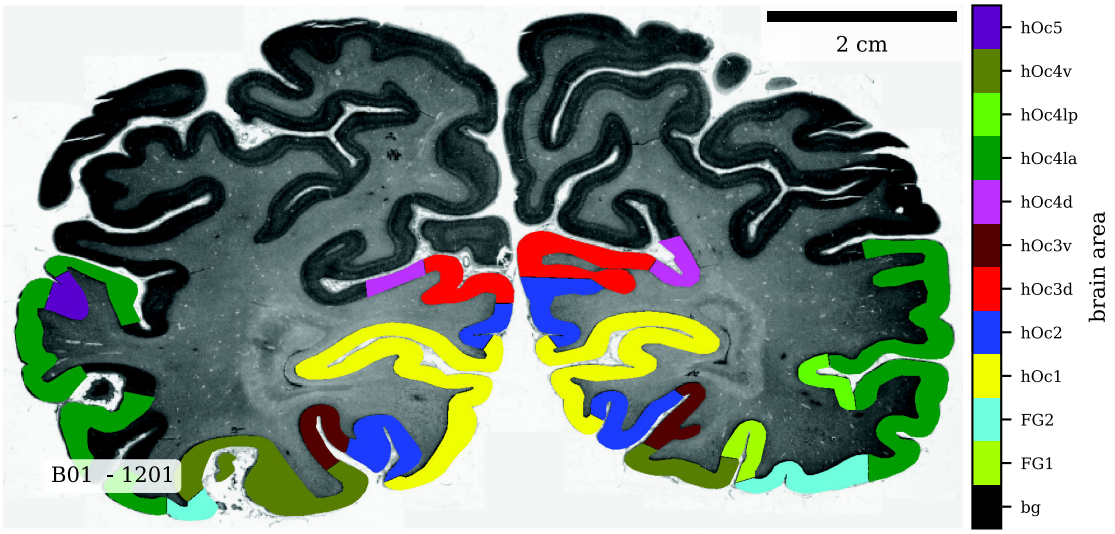



**Figure 3.5:** Annotations of cytoarchitectonic areas in section 1201 of B01. Cytoarchitectonic areas are denoted by colors. Sections are typically only partially annotated. In this section, areas in the upper half of the section are not mapped. Pixels in this region might belong to known regions that were not mapped in this particular section or to areas for which no nomenclature exists yet.

artifacts, or the local cutting angle (Section 2.1.4). Thus, sections are mostly *partially annotated*. Parts of the cortex without annotations can belong to 1) an area that was not annotated on the respective section (e.g., because a section was not used to study a specific area) or to 2) a completely new area (i.e., an area for which no nomenclature has been established yet). In addition, not all mapping projects use the same subset of brains, and all brains contain only a subset of areas. These preconditions need to be taken into account when designing algorithmic cytoarchitecture analysis methods.

Annotations of cytoarchitectonic areas are represented as contours of their outer boundaries. Contours can be rasterized to create segmentation masks of cytoarchitectonic areas (Figure 3.5). We algorithmically check contours for typical errors (e.g., overlapping and self-intersecting contours) and resolve them manually. The annotations are then visualized as overlays for the respective brain sections (similar to Figure 3.5) and visually inspected to spot and resolve remaining issues.

This work considers 113 cytoarchitectonic areas (Table 3.2). The set of all considered areas is denoted as $\mathcal{A}$. We use $\mathtt{A} \in \mathcal{A}$ as placeholder for a cytoarchitectonic area.

Figure 3.6 quantifies the annotations used for our work. In total, the dataset consists of 2219 annotated sections from 9 brains. About 75% of the areas are mapped in 6 or more brains, while about 9% are mapped in only four brains. The number of sections with at least one annotation ranges from 145 (B12) to 326 (B07)

**Table 3.2:** List of 113 cytoarchitectonic areas used in this work, grouped by their macroanatomical location. Areas are denoted by the Julich-Brain nomenclature, e.g., *hOc1* for *human occipital area 1* or *FG1* for *fusiform gyrus area 1*. Corresponding publications are given where available. For yet unpublished areas, the name of the responsible investigator at the Cécile and Oskar Vogt Institute for Brain Research is given. *hOc1,hOc2,hOc3v and hOc5 in B20 were annotated by Kai Kiwitz.

| | |
|---|---|
| **occipital lobe** | |
| hOc1, hOc2 | Amunts et al. (2000)* |
| hOc3v, hOc4v | Rottschy et al. (2007)* |
| hOc3d, hOc4d | Kujovic et al. (2013) |
| hOc4la, hOc4lp, hOc5 | Malikovic et al. (2016)* |
| hOc6 | Richter et al. (2019) |
| **parietal lobe** | |
| OP1, OP2, OP3, OP4 | Eickhoff et al. (2006) |
| 1, 2, 3a, 3b | Geyer et al. (1999) |
| 5l, 5m, 5Ci, 7PC, 7A, hIP3 | Scheperjans et al. (2008) |
| PF, PFcm, PFm, PFop, PFFt, PGa, PGp | Caspers et al. (2006) |
| hIP1, hIP2 | Choi et al. (2006) |
| hIP4, hIP5, hIP6, hIP7, hIP8, hPO1 | Richter et al. (2019) |
| **temporal lobe** | |
| FG1, FG2 | Caspers et al. (2013) |
| FG3, FG4 | Lorenz et al. (2017) |
| Te10, Te11, Te12 | Morosan et al. (2001) |
| | Rademacher et al. (2001) |
| Te21, Te22, Te3, Te4, Te5, Ti1, Ti2 | Zachlod et al. (2020) |
| **frontal lobe** | |
| 4a, 4p | Geyer et al. (1996) |
| 6d1, 6d2, 6d3 | Sigl (2018) |
| 6v1, 6v2, 6r1 | Jeanette Stangier (unpublished) |
| 6mp, 6ma | Ruan et al. (2018) |
| 11a, 11p, 13 | Henssen et al. (2016) |
| Fo4, Fo5, Fo6, Fo7 | Wojtasik (2020) |
| ifj1, ifj2, ifs1, ifs2, ifs3, ifs4 | Bradler (2015) |
| 8a, 8b, 8c, 8d | Jonas Hansel (unpublished) |
| sfs1, sfs2, fms1, mfg1 | Ariane Bruno (unpublished) |
| 44, 45 | Amunts et al. (1999) |
| OP5, OP6, OP7 | Nina Unger (unpublished) |
| OP8, OP9 | Martin Saal (unpublished) |
| **limbic lobe** | |
| 25a, 25p, s24a, s24b, s32, p24a, p24b, pv24c, pd24cd, pd24cv, p32 | Palomero-Gallagher et al. (2015) |
| **insula** | |
| Ig1, Ig2, Id1 | Kurth et al. (2010) |
| Ig3, Id2, Id3, aId1, aId2, aId3, aIa | Julian Quabs (unpublished) |
| Iad1 | Grodzinsky et al. (2020) |

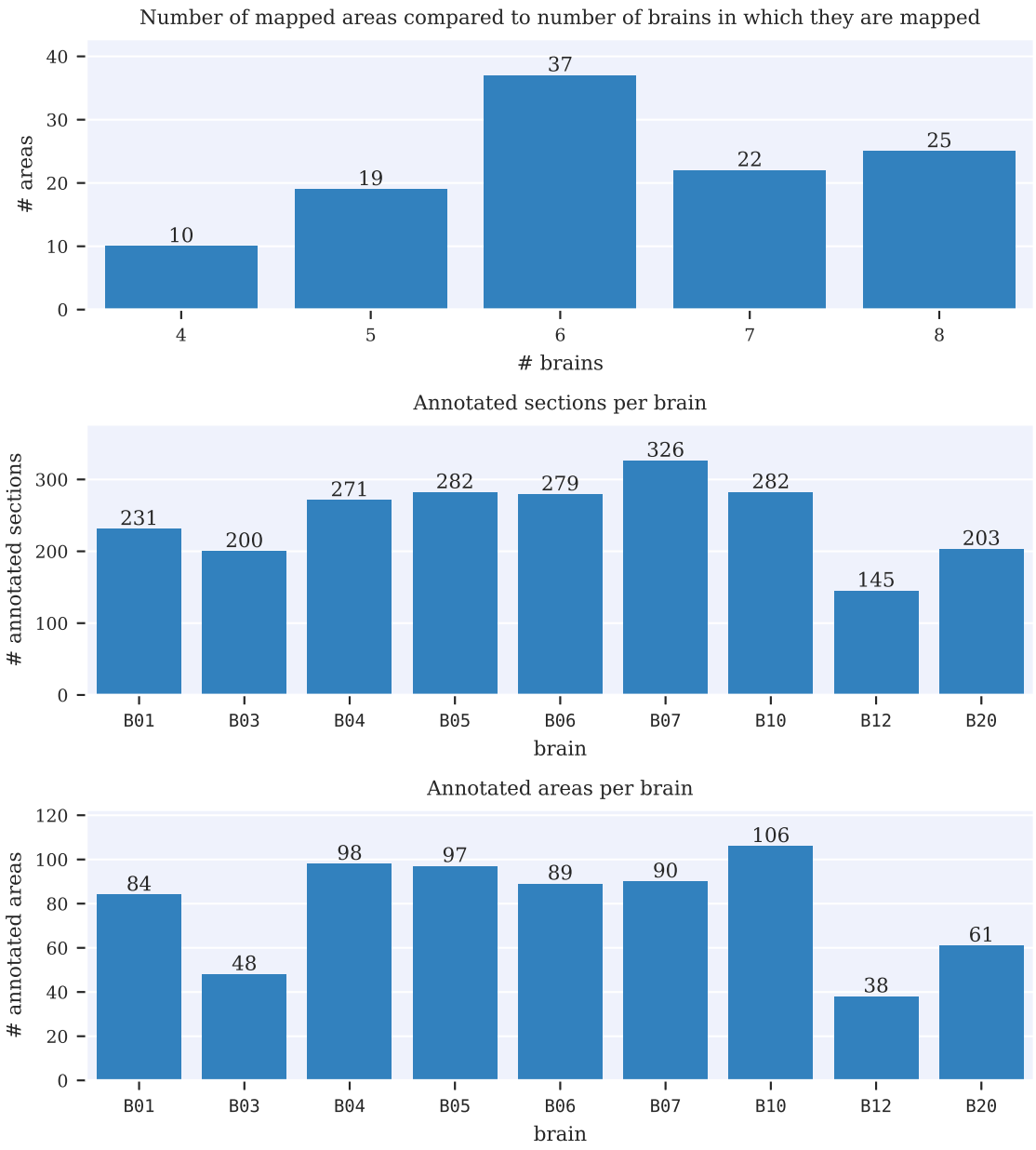



**Figure 3.6:** Statistics on annotated cytoarchitectonic areas. **Top**: Number of areas compared to the number of brains they are mapped in (e.g., 10 areas were mapped in four brains, 37 areas were mapped in six brains). **Middle**: Number of sections containing at least one annotated area across different brains. **Bottom**: Number of unique annotated areas across different brains.

sections. The number of different areas mapped in a particular brain ranges from 38 (B12) to 106 (B10).

# 4 Implementation

Tasks that can be solved with established tools on small to medium size image datasets (e.g., resizing of photos) become technically demanding when dealing with terabyte-scale image datasets, where both the quantity and the individual size of images are large (Amunts et al., 2021). Analyzing large high-resolution microscopic images puts high demands on hardware, software, and data storage. This chapter describes the used hardware (Section 4.1), software (Section 4.2), and data management concepts (Section 4.3).

## 4.1 Hardware infrastructure

### 4.1.1 High-performance computing systems

The experiments presented in this work were performed on the supercomputers JURECA (Jülich REsearch on Exascale Cluster Architectures) (Krause et al., 2018) and its successor JURECA-DC (Jülich REsearch on Exascale Cluster Architectures - Data Centric module) at the Jülich Supercomputing Centre (JSC) at FZJ, Germany (Table 4.1). JURECA was decommissioned at the end of 2020 and succeeded by JURECA-DC.

The JURECA system provided 1872 compute nodes (two Intel Xeon E5-2680 v3 Haswell CPUs per node, $2 \times 12$ cores à 2.5 GHz with hyperthreading, 128 to 512 GB memory). 75 of these nodes were equipped with two NVidia K80 GPUs each ($2 \times 4992$ CUDA cores, $2 \times 24$ GB GDDR5 memory).

The JURECA-DC system provides 768 compute nodes (two AMD EPIC 7742, $2 \times 64$ cores à 2.5 GHz with hyperthreading, 512 to 1024 GB memory, InfiniBand HDR100 interconnect). 192 of these nodes (so called *accelerator* nodes) are equipped with four NVidia A100 GPUs each ($4 \times 6912$ CUDA cores, $4 \times 432$ tensor cores, $4 \times 40$ GB HBM2e memory).

### 4.1.2 Distributed file systems

JUST (Jülich Storage Cluster) is a high-performance storage system available at the JSC (Graf et al., 2021). It is connected to the supercomputer JURECA-DC (and

**Table 4.1:** Configuration of the supercomputer systems at JSC that were used for this work. JURECA was decommissioned at the end of 2020 and succeeded by JURECA-DC.

| | JURECA | JURECA-DC |
|---|---|---|
| # nodes (w. GPUs) | 1872 (75) | 768 (192) |
| CPU | Intel Xeon E5-2680 v3 Haswell | AMD EPIC 7742 |
| CPUs/node | 2 | 2 |
| cores/CPU (virtual) | 12 (24) | 64 (128) |
| CPU clock | 2.5 GHz | 2.5 GHz |
| RAM/node | 128 GB | 512 GB |
| GPU | NVidia K80 | NVidia A100 |
| GPUs/node | 2 | 4 |
| GPU memory | $2 \times 24$ GB GDDR5 | $4 \times 40$ GB HBM2e |
| CUDA cores | $2 \times 4992$ | $4 \times 6912$ |
| tensor cores | - | $4 \times 432$ |

previously to JURECA). JUST provides a total storage capacity of about 140 PB[1] and a nominal peak bandwidth of 380 GB/s. It comprises multiple storage layers with different performance characteristics and storage capacity. JUST runs the General Parallel File System (GPFS) by IBM, a high-performance distributed file system.

The HPST (High Performance Storage Tier) is a storage layer of JUST that is of particular interest for this work. It is composed of 110 IME-140 servers by *DataDirect Network (DDN)*, offering a total capacity of approximately 2.2 PB and a total bandwidth of 2 TB/s. Compared to other storage layers of JUST, which are primarily based on mechanical storage disks, HPST is based on NVMe flash storage. This makes it well suited for file access patterns that result in suboptimal performance on mechanical disks.

## 4.2 Software

The experiments presented in this work are conducted using the specifically developed software framework ATLaS (Automatic Tissue Labeling System)[2]. ATLaS offers functionality for training deep neural networks, creating predictions, evaluating results, parallel processing of images, and visualizing different kinds of data. It was designed to address the unique challenges of applying deep learning for cytoarchitecture classification in large-scale microscopy datasets. In particular, the software is optimized to efficiently use the supercomputer resources described in Section 4.1.

[1] As of Q1 2022. JUST is frequently expanded.

[2] https://jugit.fz-juelich.de/c.schiffer/atlas

**Table 4.2:** List of important *Python* libraries used by the ATLaS framework.

| package | description | reference |
|---|---|---|
| `numpy` | numerical computing | Harris et al. (2020) |
| `scipy` | numerical computing | Virtanen et al. (2020) |
| `pandas` | tabular data analysis | Wes McKinney (2010) |
| `PyTorch` | deep learning | Paszke et al. (2019) |
| `PyTorch-Geometric` | deep learning (GNNs) | Fey et al. (2019) |
| `TensorFlow` | deep learning | Abadi et al. (2016) |
| `scikit-image` | image processing | van der Walt et al. (2014) |
| `OpenCV` | image processing | Bradski (2000) |
| `scikit-learn` | machine learning | Pedregosa et al. (2011) |
| `mpi4py` | parallel processing | Dalcin et al. (2011) |
| `numba` | parallel processing | Lam et al. (2015) |
| `h5py` | file access | Collette (2013) |
| `pytiff` | file access | Glock (2020) |
| `matplotlib` | visualization | Hunter (2007) |
| `seaborn` | visualization | Waskom (2021) |

The framework is easily extensible and builds upon popular deep learning software frameworks (Abadi et al., 2016; Paszke et al., 2019).

### 4.2.1 Software libraries

ATLaS and its accompanying software libraries are implemented in the *Python* programming language (Van Rossum, 1995). The availability of many software packages for scientific computing, machine learning, parallel processing, and image processing makes Python the language of choice for many projects involving deep learning. Table 4.2 lists the most important software libraries used by ATLaS.

### 4.2.2 Training pipeline implementation

ATLaS implements the training pipeline for deep neural networks used in this work. The pipeline encompasses continually reading data from disk, preprocessing it, and transferring it to a GPU, where it is used to repeatedly update the model parameters.

The processing pipeline is parallelized using Message Passing Interface (MPI), a software library[3] for implementing parallel applications (Message Passing Forum, 1994). MPI allows exchanging data between processes running on the same or different compute nodes, making it a flexible tool to implement parallel algorithms. It

[3]To be precise, MPI is a software *specification*. However, the term "MPI" is often used synonymously for a specific library (e.g., *OpenMPI*) that implements the MPI specification.

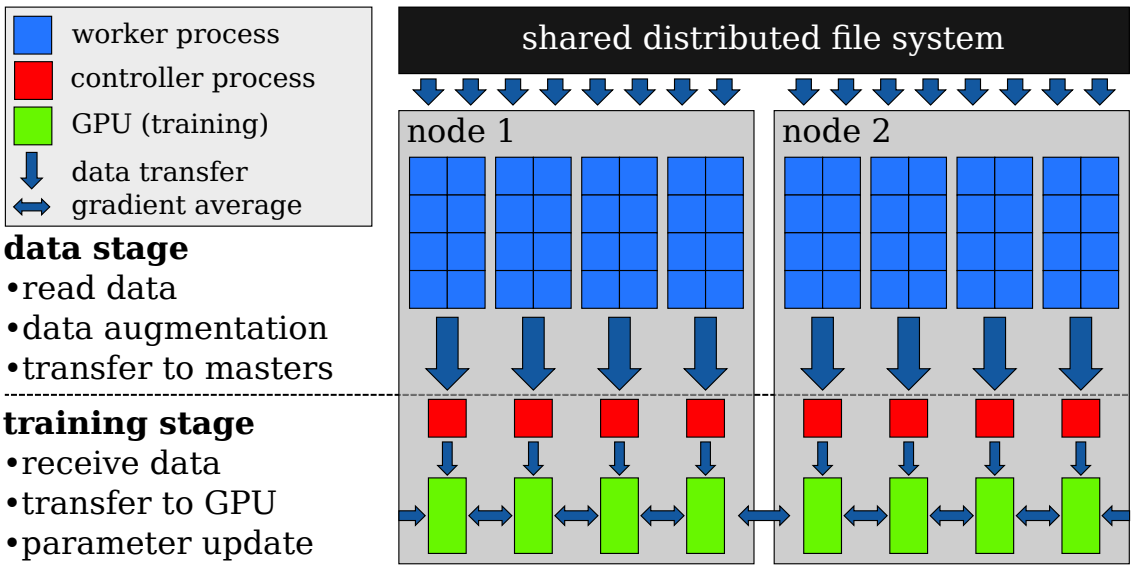



**Figure 4.1:** Data processing and training pipeline implemented by the ATLaS framework. The pipeline is parallelized with a controller-worker communication scheme that is implemented using MPI. Controller processors are each assigned one GPU and coordinate the training process of the deep neural network. The remaining processes are evenly distributed among controller processes as worker processes, designated to read data, perform data augmentation, and transfer the resulting data to their respective controller. DDL is implemented using data parallelism, so gradients are averaged across all GPUs participating in the training. This example shows two compute nodes, but the pipeline can be scaled to arbitrary numbers of nodes.

offers different communication patterns, including point-to-point communication and collective operations (e.g., one-to-all or all-to-one). Most MPI implementations are written in the C programming language but can be used from Python through the *mpi4py* library (Dalcin et al., 2011).

The pipeline can be subdivided into the *data stage* and the *training stage*. (Figure 4.1). It is parallelized using a *controller-worker parallelization scheme*[4], where several *worker* processes perform a computational task and send the results to one or multiple designated *controller* processes.

Controller processes are responsible for coordinating the training stage: They receive data from their assigned worker processes, transfer the data to a GPU, and coordinate all aspects related to the training process. The training stage uses the *PyTorch* framework (Paszke et al., 2019) and its built-in DDL module. It efficiently implements gradient averaging using a bandwidth-optimal ring-reduce algorithm (Patarasuk et al., 2009) available in the NVIDIA Collective Communication Library (NCCL)

[4] Also known as *master-worker* communication scheme.

library. The number of controller processes is set to be equal to the number of available GPUs, so each controller controls exactly one GPU.

All non-controller processes are equally distributed among the controller processes and designated as worker processes. Worker processes execute the data stage, which comprises reading training data from disk, applying data augmentation, and assembling individual images into batches before sending them to their respective controller process.

### 4.2.3 Implemented performance optimizations

The training pipeline implemented by ATLaS is optimized for high throughput, which is required to ensure optimal resource utilization and enable training in practically feasible time frames (e.g., several hours compared to days or weeks). It thus allows resource-efficient and flexible experimentation.

Wherever possible, ATLaS uses functions from established software libraries, as they are typically already highly optimized. Outgoing from an initial implementation that follows established best-practices (e.g., following `PyTorch`, Paszke et al., 2019), we identified and addressed bottlenecks encountered when processing large datasets. We use runtime and memory profilers to iteratively optimize the implementation. In the following, we list examples for implemented performance optimizations.

Image patches used for cytoarchitecture classification are considerably larger than images used in many other image analysis tasks (Russakovsky et al., 2015). The large patch size increases the computational requirements for reading, transforming, and transferring images.

ATLaS implements the storage-related performance optimizations discussed in Section 4.3. For example, it caches open file handles and associated metadata to minimize the overhead for opening files and optimize the read performance.

Image transformations (e.g., for data augmentation, Section 2.2.1) are carefully selected and optimized to use the most efficient available implementation for the used image size. If necessary, we specifically optimize and parallelize transformations for which no optimized implementations are available. For example, we implemented an optimized gamma augmentation (Section 6.1.3) based on `numba` (Lam et al., 2015).

The MPI-based implementation of the worker-coordinator communication integrates well with high-performance computing (HPC) environments, which often provide optimized software and hardware configurations for MPI-based programs. ATLaS implements MPI-based asynchronous worker-coordinator communication and thread-based asynchronous[5] CPU-to-GPU data transfer. The data transfer is thus

[5]The thread-based parallelization is not negatively affected by the *global interpreter lock (GIL)* of the cPython interpreter. The CPU-to-GPU communication releases the GIL and is performed in parallel to the main thread.

performed in parallel to data preprocessing and the actual training and induces minimal overhead.

The training pipeline often requires large amounts of *auxiliary data* to be kept in memory, for example, to determine sampling locations for image patches. Auxiliary data is typically used by all worker processes, but providing each process with its own copy of the data is technically impossible due to memory limitations. Instead, ATLaS uses the *shared memory* feature introduced in version 3 of the MPI specification. It allows sharing data among processes on the same compute node, which drastically reduces the memory requirements.

## 4.3 Data management

This section describes the implemented data management concept. A suitable data management concept enables fast access to the image data that is used for training deep neural networks. We first describe the preconditions (Section 4.3.1), which are determined by the image acquisition workflow (Section 3.1). We then detail the requirements for data management (Section 4.3.2), encountered technical challenges (Section 4.3.3), and implemented solutions (Section 4.3.4).

### 4.3.1 Preconditions

Following the histological processing steps described in Section 3.1, brain sections are scanned using high-throughput light-microscopic scanners. The scanners store each scanned image in a *BigTIFF* image file. The BigTIFF format is a variant of the Tagged Image File Format (TIFF) image format (*.tif* or *.tiff* file extension). Compared to standard TIFF, BigTIFF supports images larger than 4 GB. At highest resolution of 1 µm/px, the median size of of an image is approximately 7.5 GB. Images are stored as gray level (i.e., single-channel) images with a color depth of one byte per pixel (256 colors).

Each image contains an *image pyramid*, which consists of multiple versions of the image with decreasing resolution and size. An image pyramid allows accessing downscaled versions of the original image, facilitating visualization and analysis tasks that do not rely on the full resolution. The multi-page feature of TIFF allows storing all levels of the image pyramid in a single file. A file contains pyramid levels with 1, 4, 16 and 64 µm/px resolution, resulting in median file size of approximately 10 GB.

The image datasets considered in this work require approximately 79 TB of storage space. The many scanned sections of `B20` contribute most to these storage requirements (54 TB), while the other brains use on average 3.1 TB each. Images are stored on the file systems of the JSC (Section 4.1.2).

### 4.3.2 Requirements

The large image size prevents most algorithms from operating on whole images. Even in cases where storing whole images in system memory is technically possible (e.g., on HPC systems), most analysis methods cannot directly process images of that size. Thus, it is common practice (Ronneberger et al., 2015; Spitzer et al., 2017) to extract smaller image patches from whole images and individually feed them to an analysis method. Spitzer et al. (2017) adopt this patchwise processing scheme for their work on cytoarchitecture classification, and the methods proposed in this work also resort to performing image analysis on image patches.

Deep neural network training requires many image patches to be read from different locations and different microscopic images. If the time it takes to create, augment (Section 2.2.1), and transfer a batch of training examples (i.e., image patches) exceeds the time for processing a batch, the training process becomes *bottlenecked* by the data generation pipeline. A bottleneck situation must be avoided to optimize resource usage and enable training in reasonable time frames.

### 4.3.3 Technical challenges

**High-frequency random access for reading image patches** Extracting image patches from whole images could be efficiently implemented by reading the whole images into system memory and extracting image patches from there. However, the memory requirements make this approach technically infeasible for multi-terabyte datasets. Instead, image patches need to be read directly from files on disk.

The locations from which image patches are extracted are randomly determined during training. The resulting file access pattern appears as a *random access pattern*, which has several practical implications:

- The random access pattern prevents caching, as neither spatial nor temporal locality can be exploited to improve performance.

- Distributed file systems based on mechanical disks are not optimized for a large amount of relatively small random read operations (Pumma et al., 2017; Oden et al., 2019).

- The overhead associated with each read operation becomes overwhelming for many read operations.

In combination, the above implications of the random access pattern lead to poor read performance.

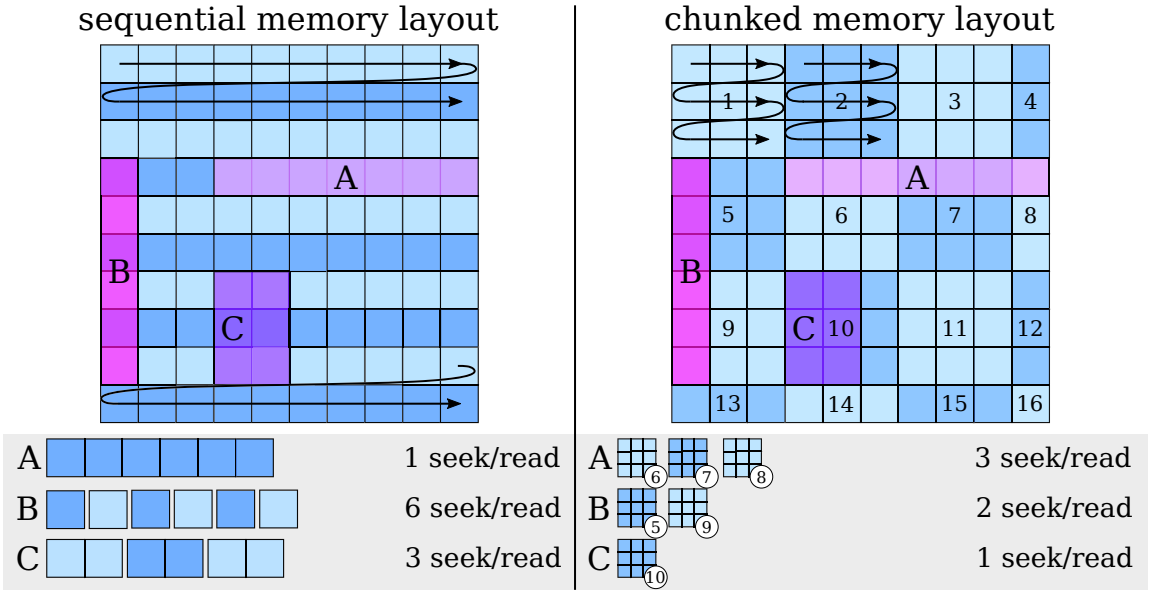



**Figure 4.2:** Storage layout for two-dimensional data with sequential and chunked memory layout. Each square represents an element of a 2D array (e.g., an image), stored in the order indicated by the arrows. The sequential layout (left) stores elements sequentially along rows. This makes access to contiguous rows efficient (A), but access to columns (B) or 2D patches (C) requires more seek and read operations. In comparison, the chunked memory layout (right) stores specific chunks of data (here $3 \times 3$ elements) sequentially. Depending on the access pattern, reading data from a chunked memory layout can require less seek and read operations, making it more efficient and flexible.

**Patchwise file access** A typical way of representing images in a machine-readable format is to store all pixel values in a contiguous memory layout (Figure 4.2), where the pixels of each row[6] are stored sequentially. This layout is efficient to read contiguous parts of a row (Figure 4.2, A) or multiple consecutive rows, as these actions can be achieved by a few file system operations (i.e., one *seek* operation to find the starting pixel and one *read* operation to read the desired number of pixels). However, it is not efficient for reading columns or 2D patches spanning multiple rows (Figure 4.2, B, C). As the data of a patch is not contiguous in memory, one pair of seek- and read-operations has to be performed for each row of a patch, resulting in many file system operations. The access pattern of an application has to be taken into account for choosing an appropriate data storage layout.

[6] In the *row major* ordering, elements of a row are stored sequentially, while in the *column major* ordering, elements of a column are stored sequentially. Row major ordering is predominantly used in the C programming language, while column major ordering is used in the Fortran programming language.

**Physical resolution mismatch** The microscopic image data has a resolution of 1 µm/px. However, existing methods for cytoarchitecture classification (Spitzer et al., 2017, 2018b), as well as the methods proposed in this work, use a resolution of 2 µm/px. Using a lower resolution represents a compromise between the provided detail and the computational resources required to process the images. Image patches at 2 µm/px resolution still provide sufficient detail on the cellular distribution, while reducing the amount of data per patch by 75% compared to 1 µm/px resolution. However, the available BigTIFF images do not contain a pyramid level with 2 µm/px resolution (Section 4.3.1). While it would be possible to downscale 1 µm/px patches on-demand during training, this would introduce a significant computational overhead and introduce a potential bottleneck in the data processing pipeline.

**Handling of many files** Patchwise training requires repeated access to a large number of files. Opening and closing each file for each read access becomes computationally infeasible, as every open and close operation incurs an overhead. This overhead becomes significant when many open and close operations are performed. Thus, file handles need to be kept open for the duration of the training. This approach avoids the negative performance impact of repeated open and close operations. However, each open file handle also occupies a certain amount of system memory and contributes to the memory footprint of an application. The memory requirements of an open file handle depend on the file type. Complex and metadata-rich file types can require several megabytes of memory per file to store the underlying data structures.

This effect is particularly severe for parallel applications, as each process has to keep all file handles open. Here, the memory requirements scale with the number of file handles and the number of processes. For example, consider an application running 64 processes on one compute node, each keeping handles of 1800 image files open[7]. If each file handle requires 1 MB of memory, the file handles across all processes require approximately 112 GB of memory. Storing the file handles thus requires a significant amount of memory.

### 4.3.4 Implemented solutions

We address the above challenges by implementing a carefully designed data format based on Hierarchical Data Format 5 (HDF5) (The HDF Group, 1997). HDF5 is a hierarchical file format for storing scientific data (e.g., time series, images, volumes, or high-dimensional arrays). Similar to a file system with files and folders, HDF5 allows to hierarchically organize datasets in different groups, to which additional attributes (e.g., metadata) can be assigned. It is thus well suited to store complex

[7] Numbers are chosen to reflect a realistic experiment from Chapter 6.

hierarchical data. In addition, HDF5 provides fine-grained control over the internally used data storage layout.

The data format is constructed in the following way: First, the highest resolution pyramid level of each microscopic image (1 µm/px) is downscaled to a resolution of 2 µm/px using cubic interpolation. This step addresses the issue of non-matching physical resolutions and ensures that training image patches can be efficiently read at the desired resolution of 2 µm/px.

The downscaled images are stored in HDF5 files. However, images belonging to the same brain are stored as datasets in a single HDF5 file rather than storing each image in a separate file. The number of brains is significantly smaller than the number of brain sections, so this approach significantly reduces the number of file handles that need to be kept open by each process. Issues arising from handling too many files are mitigated this way, as the memory overhead is significantly reduced.

HDF5 allows storing data with a chunked memory layout (Figure 4.2, right), which reduces the number of read operations to read rectangular image patches. The chunk size has to be adapted to the application's requirements, as both too small and too large chunk sizes can lead to poor performance. The images used in our experiments are stored with a chunk size of $2048 \times 2048$, which matches the typical patch size used in our experiments and has shown to work well in practice.

Using a chunked memory layout, combining multiple image files into a few large files, and reducing the image resolution improved the performance. However, the high-frequency random read access pattern regularly overloaded the available file system infrastructure (Section 4.1.2). This issue manifested as extremely variable read access times, ranging from a few milliseconds to several minutes to read a single relatively small image patch. Using the profiling tool *darshan* (Carns et al., 2011) and the Linux tool *strace*, we found that the majority of read requests that caused the high file system load were caused by a large number of small metadata read requests issued by the HDF5 library. This issue was primarily addressed by storing the HDF5 files on the HPST file system (Section 4.1.2). The flash-based HPST does not suffer from significantly degraded performance when being confronted with high-frequency random read requests.

We use the *split* storage mechanism of HDF5 to further reduce the performance impact of the metadata requests. In contrast to the default storage mechanism of HDF5, which stores both metadata (e.g., the structure of the file) and user data in the same file, the *split* mechanism stores metadata and user data in *separate* files. The small size of the metadata file (typically a few megabytes) allows the operating system to cache it in memory, significantly reducing the load on the file system. The default storage mechanism (i.e., one file for metadata and user data) results in files with a size of multiple terabytes, which are too large to be cached.

# 5 Deep learning for accelerated mapping of individual areas

In this chapter, we introduce a deep learning method for mapping individual cytoarchitectonic areas across complete series of brain sections. Instead of training a single model to classify many areas across different brains, the proposed method follows a divide-and-conquer approach to divide the brain mapping problem into easier-to-solve subtasks. Each subtask involves the classification of *a single* brain area in a *local region* of *only one* brain. One dedicated deep learning model is trained for each subtask, allowing each model to focus on local characteristics of the respective area and produce accurate predictions.

Models are trained on annotations created at approximately regular section intervals (e.g., every 120th section), which can be obtained with acceptable effort using the GLI-based brain mapping workflow (Section 2.1.4). Trained models are used to create predictions for sections without annotations in-between annotated sections to "fill the gaps". The method aims to support neuroanatomists in cytoarchitectonic brain mapping and make mapping of large section series practically feasible for the first time.

We evaluate the method by quantitative and qualitative assessment of prediction results for different areas and brain samples, partly with different tissue staining protocols (Sections 5.2.1 to 5.2.4). We examine the performance of different model architectures (Section 5.1.2), and investigate the interpretability of the trained models (Section 5.2.5). Finally, we demonstrate the practical benefit by creating high-resolution 3D maps of cytoarchitectonic areas in the BigBrain dataset, and use them to assess the anatomical consistency and plausibility of the obtained predictions (Section 5.2.6).

The method is integrated into a HPC driven web application for visualization of microscopic image data and creation of annotations (Section 5.2.7) to facilitate easy use. The application enables users to train models and observe prediction results without requiring advanced technical knowledge or expertise in deep learning.

The method reported in this chapter was published in the article "Convolutional Neural Networks for Cytoarchitectonic Brain Mapping at Large Scale" (Schiffer et al., 2021f). An analysis of features that are learned by the proposed method was

published in "Deep Learning Networks Reflect Cytoarchitectonic Features Used in Brain Mapping" (Kiwitz et al., 2020).

# 5.1 Methods

## 5.1.1 Local segmentation models

Supervised machine learning applications typically train a single model that aims to generalize well to new samples from the considered data distribution. Using a single model is practically convenient, as it can be directly applied to classify new data. Spitzer et al. (2017) adapt this established approach to segment different cytoarchitectonic areas from the visual system of multiple brains (Section 2.2.7). While promising, the results obtained using this approach are not accurate enough to support the mapping workflow in practice.

The limited performance achieved with existing approaches (Spitzer et al., 2017, 2018b) can be attributed to the high variability of the data and the complexity of the patterns defining cytoarchitectonic areas. We can assume that collecting more training data in form of annotated cytoarchitectonic areas (Section 3.3) would improve the classification performance. However, collecting a large number of additional annotations is time and labor-intensive.

Here, we instead aim to *simplify* the problem by subdividing it into easier-to-solve subtasks, each comprising the classification of a *single area* in a *few consecutive sections* of a *single brain*. The simplification is based on the following observations:

**Limited variability in local regions** Cytoarchitecture and processing-dependent features (e.g., tissue staining) vary across brains, but the variability in spatially restricted regions of a single brain (e.g., across a few consecutive sections) is limited.

**Focused classification** The patterns defining cytoarchitectonic areas are complex and ambiguous, but classifying a selected cytoarchitectonic area in a small region of a brain can be achieved without taking all subtle variations into account.

**Exploiting local macroscopic landmarks** Macroscopic features of the brain (e.g., cortical folding patterns) are generally not representative for cytoarchitecture (Amunts et al., 2007a; Amunts et al., 2015). However, they may be used as easily recognizable landmarks to aid cytoarchitecture classification in small regions of a specific brain.

We introduce local segmentation models (LSMs) to solve the defined subtasks. A LSM, denoted as $\mathrm{LSM}_{\mathrm{B}}^{s \overset{\mathrm{A}}{\leftrightarrow} t}$, is a deep neural network that is trained on a pair of

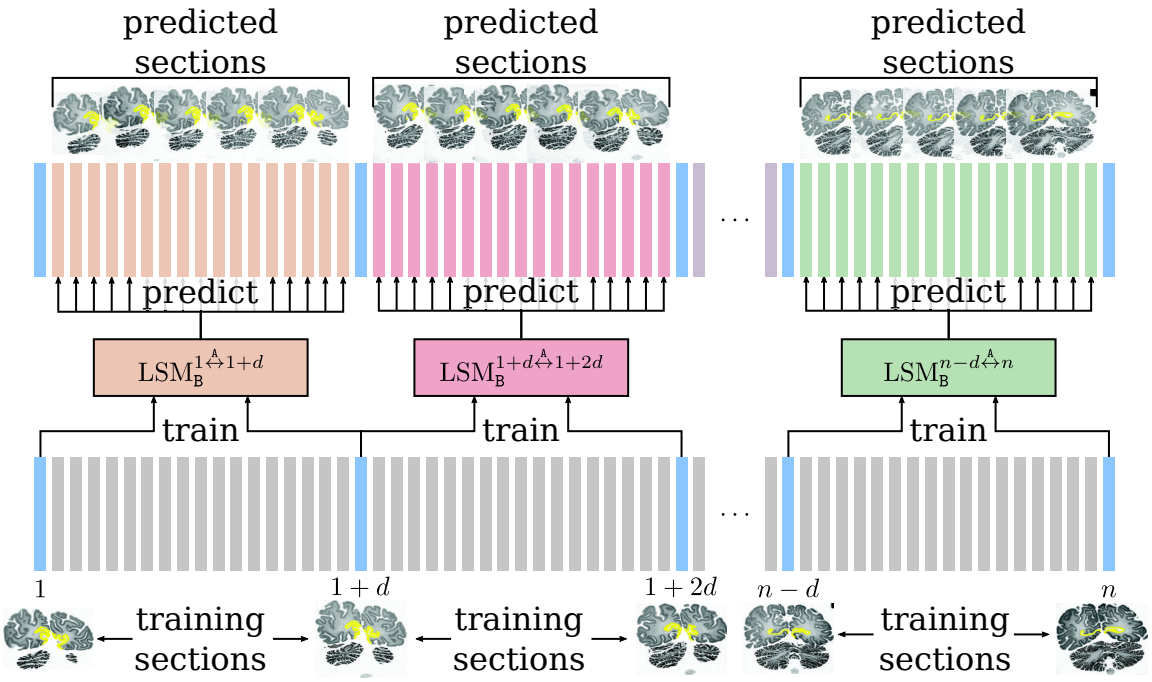



**Figure 5.1:** Illustration of the proposed LSM method for mapping large series of histological brains sections. A series of $n$ consecutive brain sections is subdivided into intervals comprising $d$ sections, where the outer sections of each interval (blue) are annotated with an area `A`. After training one LSM per interval, sections in-between the annotated sections can be mapped automatically.

spatially close *training sections* $\left\{S_s^{\mathtt{B}}, S_t^{\mathtt{B}}\right\}$ with available annotations for a *target area* `A`. We hypothesize that a LSM trained on two spatially close training sections can capture and exploit specific features of the respective region, enabling it to create accurate predictions for all sections $S_{s+1}^{\mathtt{B}}, \cdots, S_{t-1}^{\mathtt{B}}$ in-between the training sections (i.e., "fill the gaps"). Training annotations at approximately regular section intervals (e.g., every 60th-120th section) can be obtained with acceptable effort using the existing GLI-based brain mapping workflow (Sections 2.1.4 and 3.3). The larger the distance between the training sections, the more sections can be mapped automatically.

Multiple LSMs can be trained to apply the method to larger series of sections or more areas (Figure 5.1). For example, the whole extent of an area `A` across a series of sections $S_1^{\mathtt{B}}, S_2^{\mathtt{B}}, \cdots, S_n^{\mathtt{B}}$ can be captured by training a sequence of LSMs

$$\mathrm{LSM}_{\mathtt{B}}^{1\overset{\mathtt{A}}{\leftrightarrow}1+d}, \mathrm{LSM}_{\mathtt{B}}^{1+d\overset{\mathtt{A}}{\leftrightarrow}1+2d}, \cdots, \mathrm{LSM}_{\mathtt{B}}^{n-2d\overset{\mathtt{A}}{\leftrightarrow}n-d}, \mathrm{LSM}_{\mathtt{B}}^{n-d\overset{\mathtt{A}}{\leftrightarrow}n},$$

each of which can be used to segment `A` in a different section interval. Similarly, other areas can be addressed by training a dedicated set of LSMs for each area.

## 5.1.2 Dataset preparation

We model cytoarchitectonic mapping as a segmentation task: A model is tasked to assign each pixel of an image to the respective brain area. As proposed by Spitzer et al. (2017) (Section 2.2.7), segmentation is performed on image patches, as segmenting an entire brain section image is technically infeasible. The procedure used to generate image patches for training is detailed in the following.

### Auxiliary segmentation classes

Each LSM aims to segment a specific cytoarchitectonic area `A`, so modelling the task as a binary segmentation problem with a positive class "area `A`" and a negative class "not area `A`" would be straight forward. However, as already pointed out by Spitzer et al. (2017), the resulting negative class would encompass a mixture of different cytoarchitectonic areas, other tissue classes (e.g., white matter), and the background class (i.e., the microscopy slide). It is difficult to distinguish such a heterogeneous negative class from the very specific positive class.

Spitzer et al. (2017) propose to address this issue by introducing *auxiliary classes*, which further subdivide the heterogeneous negative class into semantically meaningful subclasses. Using the auxiliary classes, the task becomes a four-way segmentation problem with the classes "background" (microscopy slide), "white matter", "gray matter not belonging to `A`", and "gray matter belonging to `A`". The more distinct classes of this extended segmentation task ease the training process and improve overall performance (Spitzer et al., 2017).

Training with auxiliary classes requires segmentation masks, which we create using the tissue segmentation workflow described in Section 3.2.

### ROI-based training patch sampling

LSMs restrict training and prediction to local regions within a brain. A local region is partly defined by the training section interval $[s, t]$ on which a LSM is trained. In addition, we restrict training and prediction to tissue that immediately surrounds the area `A`.

During training, we define region of interests (ROIs) with radius $r \in \mathbb{R}^{\geq 0}$ around the annotations of `A` on the training sections (Figure 5.2). Centers for training image patches are sampled from these ROIs. $\mathcal{R}_s^{\mathtt{B}}(\mathtt{A}; r)$ denotes a ROI containing all pixels within a radius $r$ around area `A` on section $s$ of brain `B`.

Since the training of a LSM is restricted to a ROI around an area `A`, good prediction results can only be expected within equivalent ROIs on sections in-between the

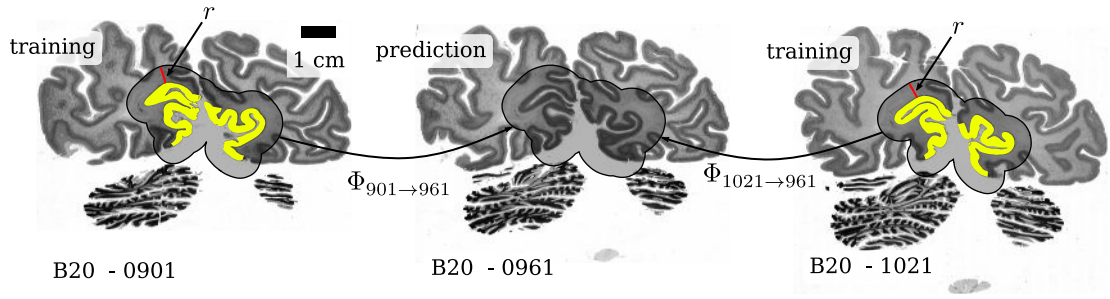



**Figure 5.2:** ROI computation for LSM training of area `hOc1` (yellow). On the training sections (here 901 and 1021 of `B20`), ROIs are defined by including all pixels within a specified radius (here $r = 5\,\text{mm}$) around area `A` . Training ROIs are linearly transformed to sections in-between (here 961) to estimate approximately corresponding ROIs for prediction.

training sections. We compute approximately equivalent ROIs

$$\tilde{\mathcal{R}}_u^{\mathtt{B}}(\mathtt{A}; r, s, t) = \Phi_{s\to u}\big(\mathcal{R}_s^{\mathtt{B}}(\mathtt{A}; e_r\, r)\big) \cup \Phi_{t\to u}\big(\mathcal{R}_t^{\mathtt{B}}(\mathtt{A}; e_r\, r)\big)\,. \tag{5.65}$$

for sections $s < u < t$ in-between the training sections $s$ and $t$ by linearly transforming (Section 3.2) the known ROIs from the training sections onto $u$. The *enlargement factor* $e_r \in \mathbb{R}^{\geq 0}$ enlarges the ROIs before transforming them, which accounts for potential inaccuracies in the transformation step. In our experiments, we use $r = 5\,\text{mm}$ and $e_r = 1.05$.

### Balanced training patch sampling

Center points for training image patches are sampled from the ROIs defined on the two training sections. The probability of a pixel to be selected as a patch center is inversely proportional to the frequency of its assigned class (Figure 5.3). Consequently, pixels belonging to a class that rarely occurs in the ROIs are selected more often than pixels belonging to a more frequent class. This approach addresses the class imbalance, which is a common issue in many machine learning applications. When confronted with an imbalanced dataset, machine learning models tend to focus on frequent classes. This issue is particularly relevant for applications where rare classes are of particular interest (e.g., detecting rare diseases). The balanced sampling procedure corrects class imbalances by oversampling rare classes and thus leads to improved performance in cases where the area `A` is small compared to the surrounding ROI.

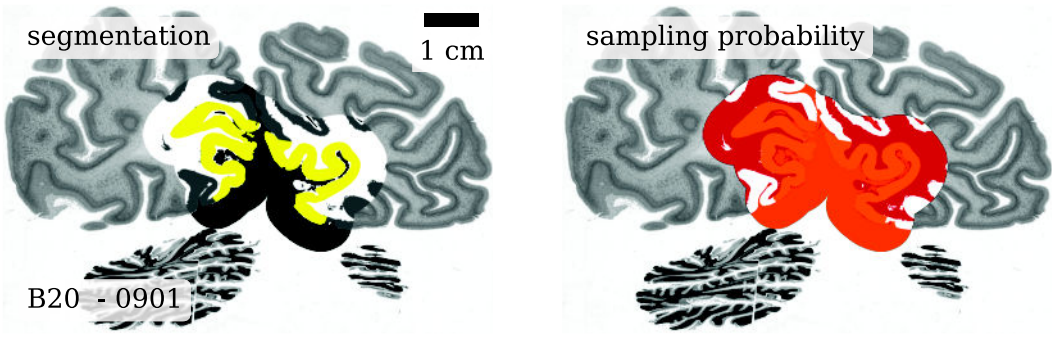



**Figure 5.3:** Sampling probabilities for drawing training image patches on section 901 of B20. **Left**: Groundtruth segmentation mask within a ROI around hOc1 (black: background, white: white matter, gray: gray matter, yellow: hOc1). **Right**: Sampling probabilities for each pixel within a ROI around hOc1. Brighter pixel values indicate a higher sampling probability for the respective pixel. This particular ROI includes less gray matter than white matter, so we assign a higher sampling probability to pixels in the gray matter. The probability of sampling a pixel of a specific class is identical for all classes (25%), resulting in a balanced training dataset.

### 5.1.3 Data augmentation

Before data augmentation operations are applied, image patches are rotated such that their upper border points to the superior (upper) part of the brain. This normalization is also applied during the prediction phase.

During training, images are randomly rotated by a small angle $\theta \sim U\left[-\frac{\pi}{4}, -\frac{\pi}{4}\right]$. Pixel intensities $x \in [0, 1]$ of a patch are randomly augmented according to $\alpha x^{\gamma} + \beta$ with parameters $\alpha \sim U[0.9, 1.1], \beta \sim U[-0.2, 0.2]$ and $\gamma \sim U[0.8, 1.214]$. The parameters of the intensity transformation are chosen to reflect natural variations observed in the data.

### 5.1.4 Training parameters

Each LSM is trained for 3000 training iterations using categorical cross-entropy loss (Equation 2.14). The terms of the loss are weighted inversely proportional to the share of the respective class within the training ROIs (Section 5.1.2). Optimization is performed using the SGD (Section 2.2.1) optimizer with Nesterov momentum (Sutskever et al., 2013), momentum $\mu = 0.9$, and a total batch size of $b = 64$ image patches (16 image patches per GPU with 4 GPUs). A weight decay of $\omega = 0.0001$ is applied to all non-bias parameters. The initial learning rate is set to $\lambda = 0.04$ and halved after 1000, 1400, 1800, 2200, and 2600 training iterations. All LSMs are trained using the same hyperparameters.

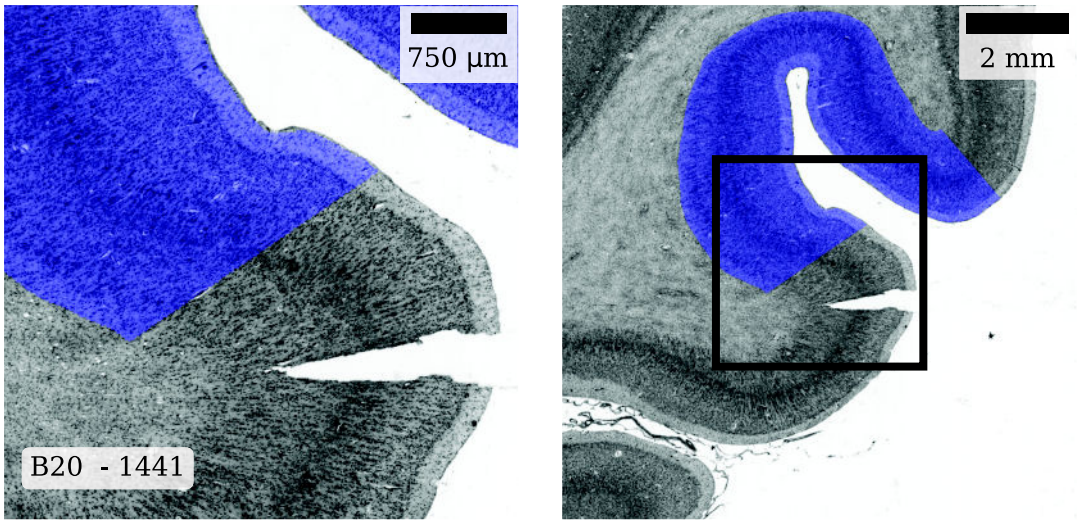



**Figure 5.4:** Multi-resolution input image patches processed by the MS U-Net architecture. **Left**: High-resolution image patch with a size of $2025 \times 2025\,\mathrm{px}^2$ at $2\,\mu\mathrm{m/px}$, which captures high-resolution image features down to the level of individual cells. **Right**: Low-resolution image patch with a size of $628 \times 628\,\mathrm{px}^2$ at $16\,\mu\mathrm{m/px}$ , which enables the model to capture macroscopic features like the folding pattern of the brain surface. Area `hOc2` is depicted in blue.

Training a LSM uses one compute node of the JURECA-DC supercomputer at JSC. Training is distributed using data-parallel DDL. The training pipeline is implemented using ATLaS (Section 4.2.2).

### 5.1.5 Network architectures

We evaluate different model architectures for the LSMs (Figure 5.5). We adopt the modified U-Net (Ronneberger et al., 2015) model (Section 2.2.3) from Spitzer et al. (2017) as a baseline. The model has shown to be well suited for cytoarchitecture segmentation. Compared to the original U-Net proposed in Ronneberger et al. (2015) (Section 2.2.3), this modified U-Net accounts for the large image patch size that is required to capture detailed cytoarchitectonic features. The model uses two convolutional layers and a pooling layer as additional initial layers. The first convolutional layer has a large filter size of $5 \times 5$ with stride 4 to allow processing of large image patches. No padding is used to avoid boundary artifacts. The model receives image patches with a size of $2025 \times 2025\,\mathrm{px}^2$ at $2\,\mu\mathrm{m/px}$ ($4.05 \times 4.05\,\mathrm{mm}^2$) and outputs segmentations with a size of $64 \times 64\,\mathrm{px}^2$ at $16\,\mu\mathrm{m/px}$ ($1.024 \times 1.024\,\mathrm{mm}^2$). We refer to this architecture as high-resolution U-Net (HR U-Net), as it is designed to extract high-resolution image features.

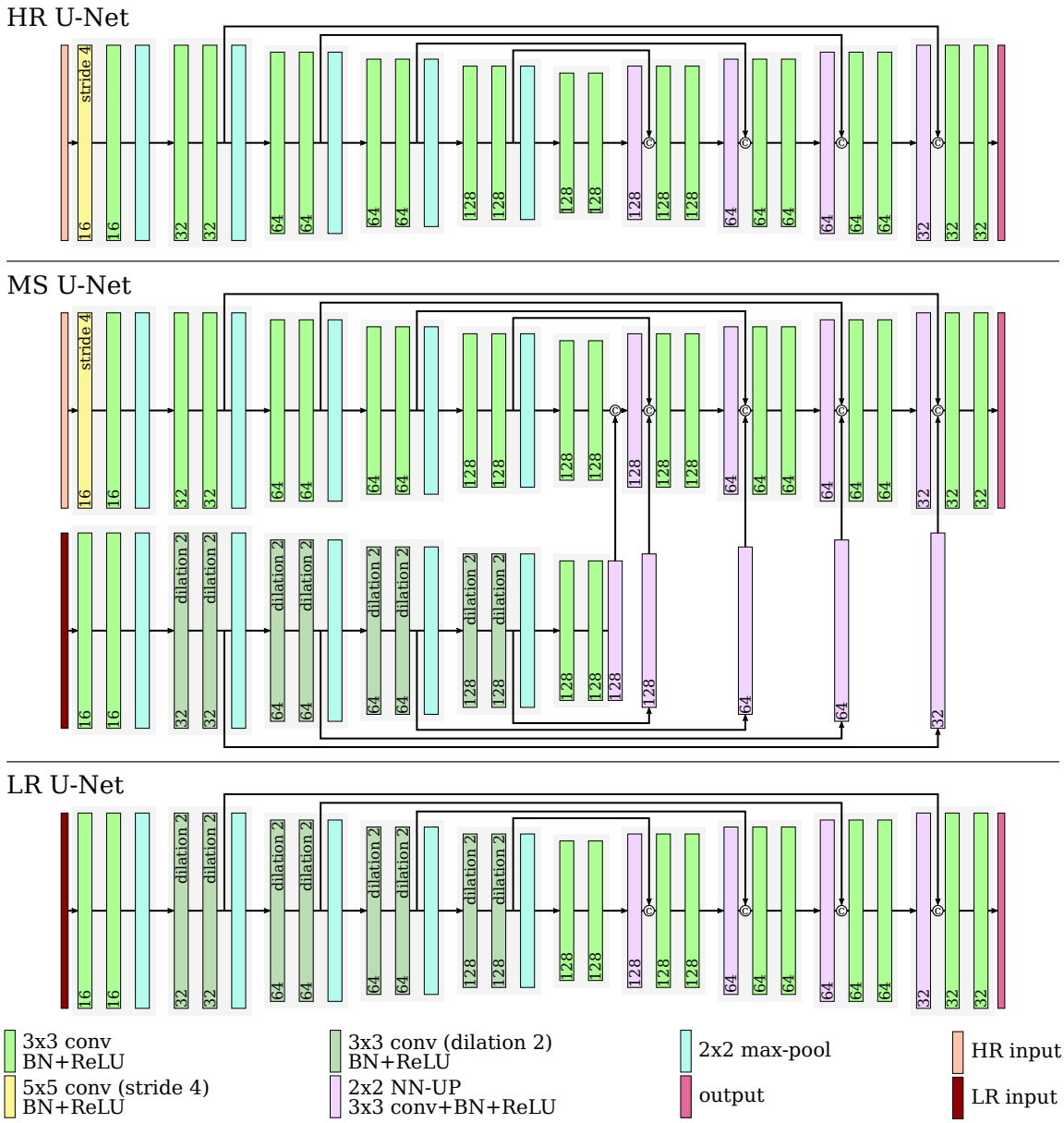



**Figure 5.5:** Architectures for LSM models. The HR U-Net model receives high-resolution image patches to capture fine details of cytoarchitecture. The LR U-Net model receives larger image patches with a lower resolution that enable macroscopic tissue features to be captured. The MS U-Net model combines the encoders of HR U-Net and LR U-Net to exploit high-resolution and macroscopic image features simultaneously. Numbers in the blocks denote the number of filters in the respective convolutional layer. The encircled "c" denotes concatenation along the feature dimension.

LSMs can rely on macroscopic features (e.g., local folding patterns) to classify cytoarchitectonic areas in local regions (Section 5.1.1). However, the image patches processed by HR U-Net are too small to capture such macroscopic features. We therefore propose the low-resolution U-Net (LR U-Net), a U-Net variant designed to exploit macroscopic image features for cytoarchitecture segmentation. The LR U-Net model receives image patches with a size of $756 \times 756\,\text{px}^2$ at $16\,\mu\text{m/px}$ ($12.096 \times 12.096\,\text{mm}^2$) and outputs segmentations with a size of $64 \times 64\,\text{px}^2$ at $16\,\mu\text{m/px}$ ($1.024 \times 1.024\,\text{mm}^2$). The lower resolution and larger field of view enable LR U-Net to extract macroscopic image features. LR U-Net uses dilated convolutional layers in all but the first two layers of the encoder to increase the field of view.

Finally, we propose the multi-scale U-Net (MS U-Net) architecture, which combines the advantages of both HR U-Net and LR U-Net using a multi-scale approach. The MS U-Net is composed of two encoders, one for high-resolution image patches and one for low-resolution image patches (Figure 5.4). The high-resolution encoder is structurally equivalent to the encoder of the HR U-Net model, while the low-resolution encoder is structurally equivalent to the encoder of the LR U-Net architecture. The outputs of both encoders are merged using concatenation. Skip-connections transfer feature maps from both encoders over to the decoder. The high-resolution encoder receives image patches with a size of $2025 \times 2025\,\text{px}^2$ at $2\,\mu\text{m/px}$ ($4.05 \times 4.05\,\text{mm}^2$), while the low-resolution encoder receives image patches with a size of $628 \times 628\,\text{px}^2$ at $16\,\mu\text{m/px}$ ($10.048 \times 10.048\,\text{mm}^2$). MS U-Net outputs segmentations with a size of $64 \times 64\,\text{px}^2$ at $16\,\mu\text{m/px}$ ($1.024 \times 1.024\,\text{mm}^2$).

The architecture of the high-resolution encoders of HR U-Net and MS U-Net is identical to the one used for the SSL task proposed in Spitzer et al. (2018b) (Section 2.2.7). Spitzer et al. (2018b) demonstrate that pre-initializing the parameters of a model for cytoarchitecture segmentation with parameters trained by their proposed SSL task can improve segmentation performance. We therefore pre-initialize the high-resolution encoders of HR U-Net and MS U-Net from weights trained with the SSL method from Spitzer et al. (2018b).

## 5.2 Results

We evaluate the method's performance for different cytoarchitectonic areas, brains, model architectures, and training paradigms.

The GLI-based brain mapping workflow (Section 2.1.4) typically considers every 60th (1.2 mm distance) section of a brain (Section 3.3). Therefore, we use every 120th section (2.4 mm) for training a LSM (Figure 5.6). In most cases, this approach leaves us with one annotated section centered between each pair of training sections. The left-out *evaluation sections* are used for performance evaluation.

The AAHA dataset is comprised of thicker sections than B01 and B20 (50 µm vs. 20 µm). Here, we adapt the distance between the sections to ensure that the physical distance between the training sections is similar for all brain samples.

Results are evaluated based on the F1-score for the respective area A (Equation 2.19). The auxiliary classes that are introduced to improve the training behavior (Section 5.1.2) are not considered for the F1-score computation, as only the segmentation of the area A is practically relevant. As described in Section 5.1.2, predictions for an arbitrary section $s < u < t$ are created within a ROI $\tilde{\mathcal{R}}_u^{\mathtt{B}}(\mathtt{A}; r, s, t)$ (Equation 5.65, Figure 5.2), which is computed based on a pair of surrounding training sections $s$ and $t$.

### 5.2.1 Local vs. global segmentation models

We evaluate the LSM models by comparing them to models that are trained on *all* sections for which a specific brain area is annotated. We call these models global segmentation models (GSMs). One GSM is trained on the union of *all* training sections available for a specific area in a specific brain. Consequently, only one GSM is trained per brain area, while multiple LSMs are trained per brain area. GSMs are trained using the same training parameters as the LSMs (Section 5.1.4).

We compare the performance of LSMs and GSMs on B20. We consider visual areas hOc1, hOc2, hOc3v and hOc5, frontal opercular areas OP5, OP6 and OP7, Broca's areas 44 and 45, areas hIP5, hIP6, hIP7 and hIP8 from the intraparietal sulcus, supplementary motor area 6mp, pre-supplementary motor area 6ma, and premotor areas 6d1, 6d2, and 6d3. Corresponding references for all areas are listed in Table 3.2.

We repeat all experiments in B20 using the HR U-Net, LR U-Net, and MS U-Net model architectures (Section 5.1.5) and compare their performance with LSM and GSM training.

Figure 5.7 shows median F1-scores obtained by LSMs, and GSMs for each of the considered brain areas. Table 5.1 shows mean, median, and standard deviation of the F1-scores across all brain areas. Statistics are computed across evaluation sections. In both cases, results are broken down based on the training paradigm (LSM, GSM) and the model architecture (HR U-Net, LR U-Net, MS U-Net).

The statistics in Table 5.1 show that LSMs generally outperform the corresponding GSMs using the same model architecture. For both LSM and GSM training, LR U-Net achieves higher median scores than HR U-Net, and MS U-Net achieves higher median scores than LR U-Net. LSM models with either LR U-Net or MS U-Net architecture show lower standard deviation, indicating more robust performance across areas. Overall, best performance is achieved by the LSM models with MS U-Net architecture.

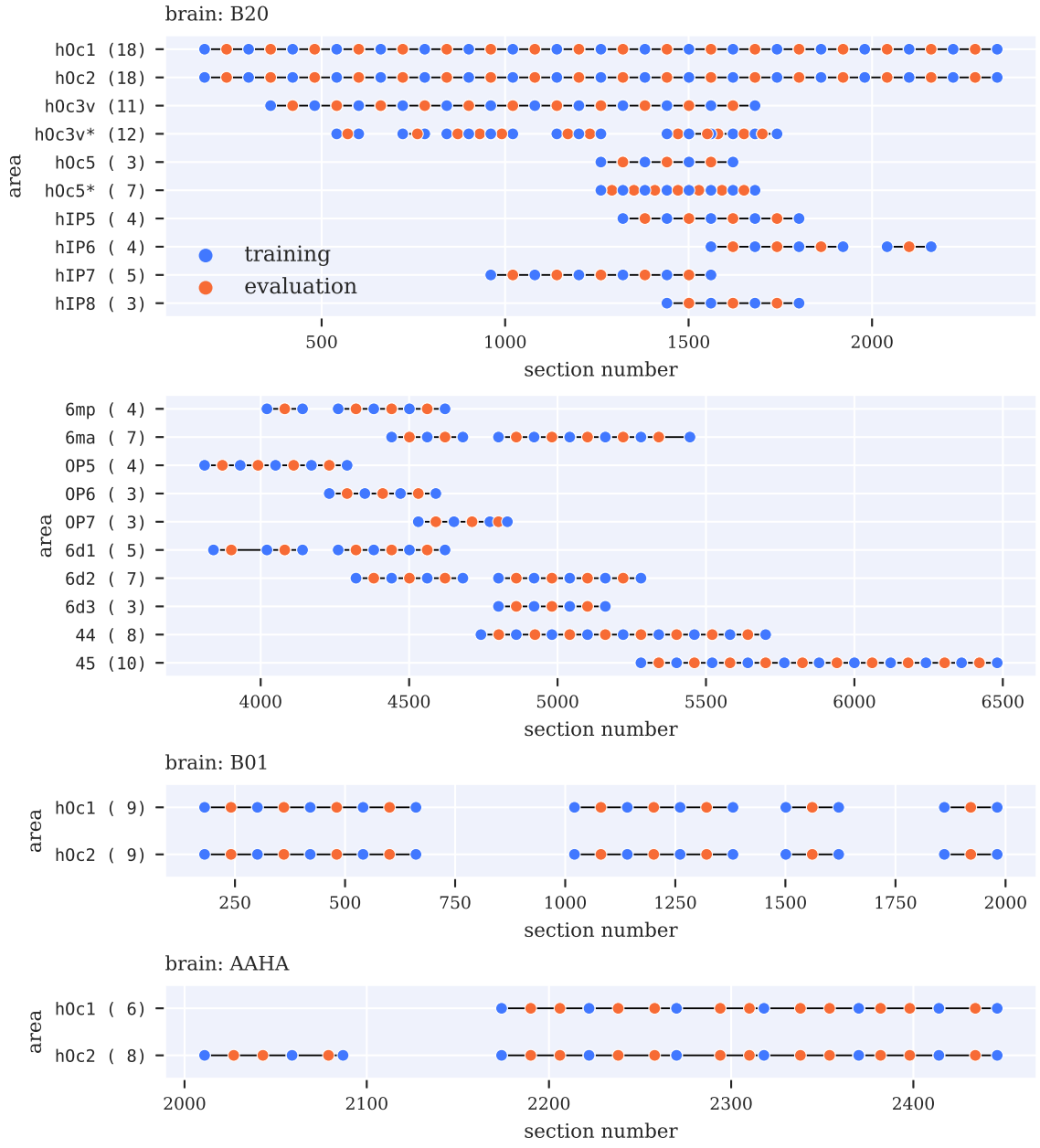



**Figure 5.6:** Training and evaluation sections used in our experiments. For each area, the diagram indicates which sections are used for training (blue dots) and evaluation (orange dots). A black line connecting two training sections indicates that the respective interval was used to train a LSM. The number next to each area name specifies the number of LSMs trained for the respective cytoarchitectonic area. Experiments `hOc3v`$^*$ and `hOc5`$^*$ use a reduced distance between training sections (Section 5.2.3). Note that the AAHA dataset uses a different section numbering scheme, so section ranges for `hOc1` and `hOc2` do not align with `B01` and `B20`.

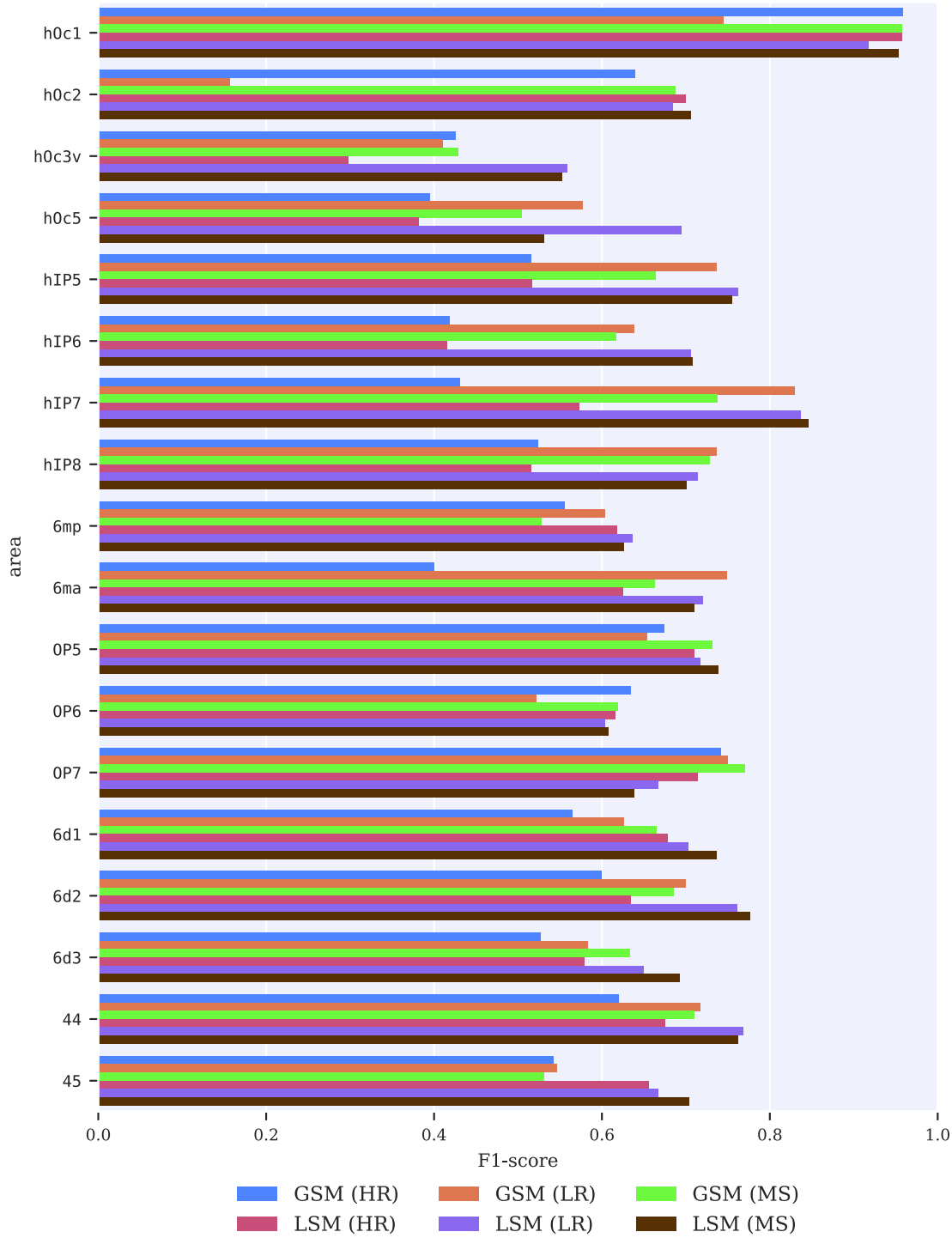



**Figure 5.7:** Median F1-scores obtained by different model architectures (HR U-Net, LR U-Net, MS U-Net) and training paradigms (LSM, GSM) per brain area in B20.

| | median | mean | std |
|---|---|---|---|
| GSM (HR) | 0.566 | 0.582 | 0.217 |
| GSM (LR) | 0.602 | 0.574 | 0.220 |
| GSM (MS) | 0.670 | 0.648 | 0.211 |
| LSM (HR) | 0.658 | 0.619 | 0.214 |
| LSM (LR) | 0.708 | 0.696 | 0.181 |
| LSM (MS) | 0.718 | 0.710 | 0.187 |

**Table 5.1:** Median, mean and standard deviation of F1-scores obtained by different model architectures (HR U-Net, LR U-Net, MS U-Net) and training paradigms (LSM, GSM) across all areas in B20 (Figure 5.7).

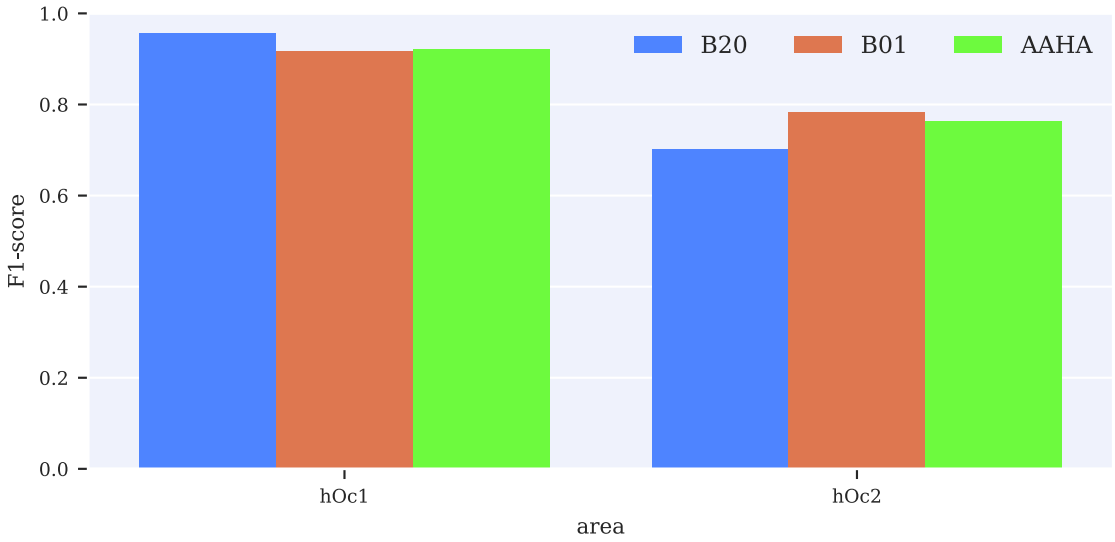



**Figure 5.8:** Median F1-scores obtained by LSMs with MS U-Net architecture on brains B20, B01, and the AAHA dataset.

The impact of the training paradigm and the model architecture varies between areas. For example, LSM models with LR U-Net architecture achieve significantly higher scores than GSM models with the same architecture for areas hOc1, hOc2, hOc3v, and hOc5. For other areas (e.g., hIP5, hIP7, hIP8), the difference between LSM and GSM training is small. In general, LSMs tend to outperform GSMs for larger areas (i.e., areas that cover many sections), while their performance is comparable for smaller areas. Larger areas show a higher variation of macroscopic properties (e.g., cortical folding patterns). The results thus suggest that the subdivision approach of LSMs allows them to better capture these macroscopic variations and classify larger areas more reliably than GSMs.

### 5.2.2 Performance on different brains

The applicability of the method to different brain samples is evaluated by investigating the performance of LSMs for areas hOc1 and hOc2 in B01, B20, and in the AAHA

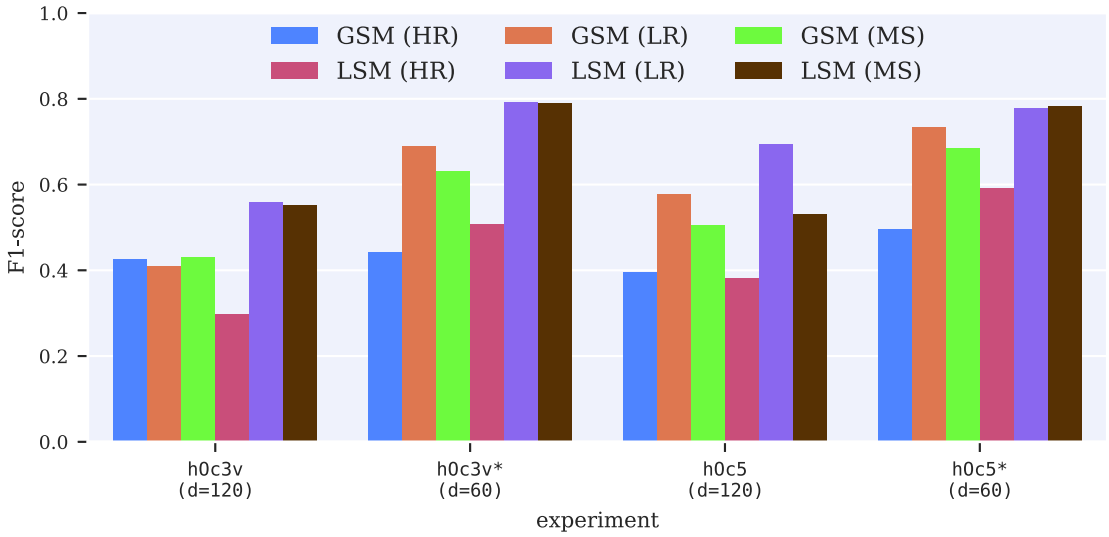



**Figure 5.9:** Median F1-scores obtained by different model architectures (HR U-Net, LR U-Net, MS U-Net) and training paradigms (LSM, GSM) for areas `hOc3v` and `hOc5` in `B20`. Models trained for experiments `hOc3v` and `hOc5v` use approximately every 120th section for training, while models in experiments `hOc3v`* and `hOc5v`* use approximately every 60th section for training.

dataset. Models trained in these experiments use the MS U-Net architecture. Brains `B01` and `B20` have undergone similar histological processing protocols (Section 3.1), while sections from the AAHA dataset were acquired using different processing steps (most notably a different staining technique, Section 2.1.3). The experiments enable us to assess the applicability of the method to new brain subjects and processing protocols.

Figure 5.8 shows median F1-scores for areas `hOc1` and `hOc2` in brains `B20`, `B01`, and AAHA. For both areas, the scores obtained in all three brains are on a comparable level, with the performance of `hOc1` being generally higher than for `hOc2`. The comparable performance across differently prepared brains indicates that the method is applicable to new brain samples without requiring specific tuning of the method's hyperparameters.

### 5.2.3 Influence of annotation density

The LSM training allows to adjust the distance between the training sections to adaptively provide additional training annotations and thereby improve prediction performance. We investigate the benefit of adding more annotations by conducting experiments for areas `hOc3v` and `hOc5` with a reduced distance of 60 sections (1.2 mm) between training sections in `B20`. These experiments are indicated by `hOc3v`* and

hOc5*. To evaluate the prediction results, additional annotations for hOc3v and hOc5 on approximately every 30th section were collected[1].

Figure 5.9 shows the median F1-scores for areas hOc3v and hOc5 in B20 obtained using LSM and GSM training with different model architectures (HR U-Net, LR U-Net, MS U-Net). Experiments are conducted with the default distance of 120 sections between training sections and a reduced distance of 60 sections.

For both LSM and GSM training, the performance improves when reducing the training distance. For LSM models, this can be attributed to the reduced variability in the data, while for GSM the larger number of available training sections explains the improved performance. In comparison to the LR U-Net and MS U-Net architectures, the performance improvement with the HR U-Net architecture is relatively small when reducing the distance between training sections. The results indicate that reducing the distance between training sections is particularly useful for LR U-Net and MS U-Net models, which can make use of macroscopic image features. In line with the results for the other B20 areas (Section 5.2.1), LSM models with either LR U-Net or MS U-Net architecture obtain the best performance.

### 5.2.4 Qualitative results

We assess the results by visual inspection of predictions on the validation sections. A selection of representative image patches and corresponding segmentations from each area in B20 is shown in Figure 5.10. The segmentations are obtained using LSMs with MS U-Net architecture. Results for hOc3v and hOc5 are obtained from experiments hOc3v* and hOc5* (Figure 5.6). The example patches are selected to give a realistic impression of the results and typically encountered errors. They do not necessarily represent the best segmentation obtained for the respective area.

The results show that the models correctly identify a large portion of each area. Errors commonly occur at the border between areas, i.e., when one area transitions into the respective adjacent area. This effect is expected, as boundaries between adjacent areas can often not be localized with absolute certainty. It can thus be assumed that both the model predictions and the reference annotations are less precise close to areal borders.

Regions where the tissue is cut obliquely relative to the pial boundary (Section 2.1.4) are another common problem (e.g., Figure 5.10 I,J,L,M,N,O,R). In these regions, the laminar organization of the cortex is not completely visible, which often prevents the exact identification of cytoarchitectonic areas. The method handles obliquely cut regions well in some cases (e.g., Figure 5.10 I,J,M,O,R), but fails in

[1]Thanks to Kai Kiwitz (Cécile and Oskar Vogt Institute for Brain Research, Heinrich-Heine-University Düsseldorf (HHU)) for providing annotations of areas hOc1, hOc2, hOc3v and hOc5 in B20 that were used to evaluate the method.

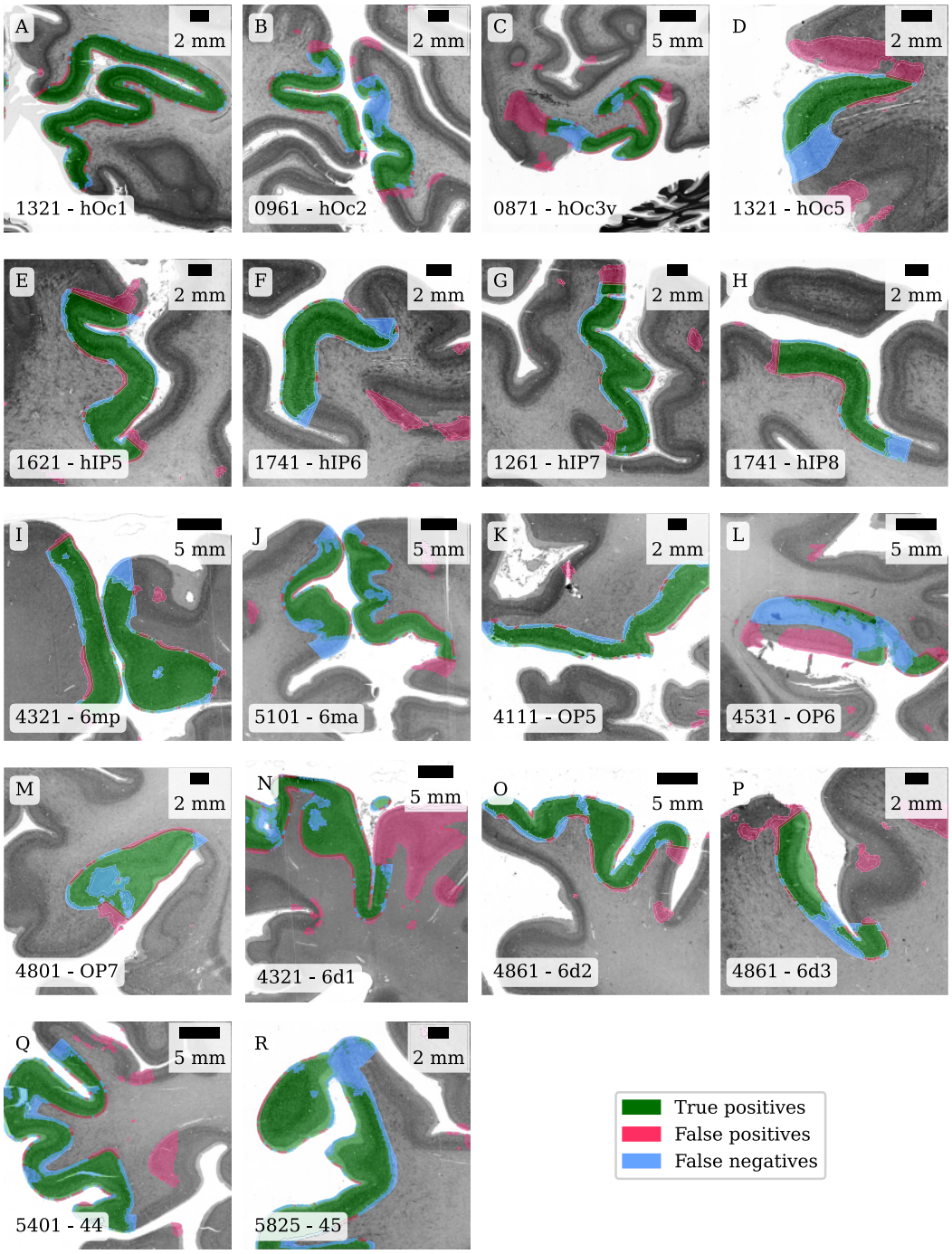



**Figure 5.10:** Image patches and corresponding segmentations obtained by LSMs with MS U-Net architecture from each considered brain area in B20. Colors indicate correctly classified pixels (green), false positive predictions (red) and false negative predictions (blue). Image patches are selected to give an overview of representative results and typical error. Segmentations of `hOc3v` and `hOc5` are obtained by models trained with reduced training distance (`hOc3v`$^*$ and `hOc5`$^*$ in Figure 5.6)

other cases (e.g., Figure 5.10 L,N). Since cytoarchitecture is only partially captured in the shown examples, we assume that morphological features recognized by the low-resolution encoder of the MS U-Net help to disambiguate some of the obliquely cut regions.

Furthermore, we found that segmentation performance tends to be lower at the extremal ends of an area (i.e., the first or last sections containing an area). This problem is also known from GLI-based brain mapping (Section 2.1.4), where it becomes increasingly challenging (and sometimes impossible) to map the extremal ends of certain brain areas. Challenges arise because the size of areas within the sectioning plane shrinks towards the extremal ends, making them challenging to delineate manually or algorithmically.

### 5.2.5 Model interpretability: What do models learn?

Understanding and interpreting how a trained deep neural network creates predictions can be important to assess its performance, estimate generalization capabilities, and identify shortcomings. Compared to some other machine learning methods (e.g., linear regression or decision trees), their hierarchical layer structure and large number of parameters make deep neural networks more difficult to understand and interpret. Some methods aim to understand the internal decision processes in trained deep neural networks by visualizing extracted features (Zeiler et al., 2014; Springenberg et al., 2015; Selvaraju et al., 2017) or model parameters (Bach et al., 2015).

To gain some intuition about how the MS U-Net architecture makes predictions, we adopt a variant of the occlusion approach proposed in Zeiler et al. (2014). The main idea is to investigate how the predictions of a model change when certain parts of the input are occluded (e.g., by setting certain pixels to zero). This can help to identify the parts of an image that are most relevant for a prediction. We adapt this idea by occluding (i.e., set to zero) either the high- or low-resolution input image of a LSM with MS U-Net architecture. This occludes the microanatomical or macroanatomical information in the input signal, respectively. We compare the results to those obtained when using intact inputs.

Figure 5.11 shows segmentations of `hOc2` on section 1441 of `B20` obtained by $\mathrm{LSM}_{\mathtt{B20}}^{1381 \overset{\mathtt{hOc2}}{\leftrightarrow} 1501}$ when using different combinations of input image patches. When restricted to lower resolution image patches (*LR only*), the model only approximately localizes the area, but fails to capture details. Similarly, the model fails to localize the position of `hOc2` when only higher resolution image patches (*HR only*) images are provided. When both resolutions are available (*LR & HR*), the model accurately segments the area.

One possible explanation for the observed behavior could be that the model approximately localizes the area using lower resolution information and identifies detailed

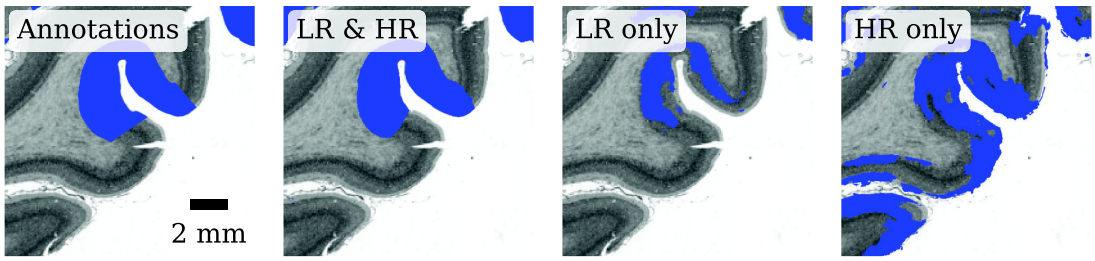



**Figure 5.11:** Segmentations of hOc2 on section 1441 of B20 obtained by $\text{LSM}_{\text{B20}}^{1381 \overset{\text{hOc2}}{\leftrightarrow} 1501}$ when specific input resolutions are occluded (Zeiler et al., 2014). *LR only* and *HR only* show the segmentation results when only the low- or the high-resolution encoder of the MS U-Net architecture receive proper input images, while the respective other encoder receives an image full of zeros. Area hOc2 is depicted in blue.

borders using higher-resolution information. This would correspond with our motivation for the MS U-Net architecture. However, while this experiment provides insights into one specific model, it does not necessarily allow general statements about the decision mechanism of other models.

Kiwitz et al. (2020) performed a comprehensive study of the features that trained LSM models learn to extract from image patches. They visually inspected the features maps of several trained LSM models and grouped them into semantically similar groups. They found that many learned features resemble cytoarchitectonic properties that are used in GLI-based brain mapping (Section 2.1.4).

### 5.2.6 High-resolution 3D maps in the BigBrain dataset

We demonstrate the practical utility of the method by using it to compute high-resolution 3D cytoarchitectonic maps (Figure 5.12) in the BigBrain (Amunts et al., 2013) dataset. A 3D cytoarchitectonic map allows studying the 3D structure of an area (e.g., its shape, volume, and surface area). For example, these structural properties can be used to study the variability of an area across different brains (Amunts et al., 2000).

Creating a cytoarchitectonic map from histological brain sections requires annotating a contiguous series of sections and assembling a consistent 3D model from these individual annotations. We use LSMs to segment areas hOc1, hOc2, hOc3v, and hOc5 in all sections of brain B20. We then reuse the existing reconstruction workflow of

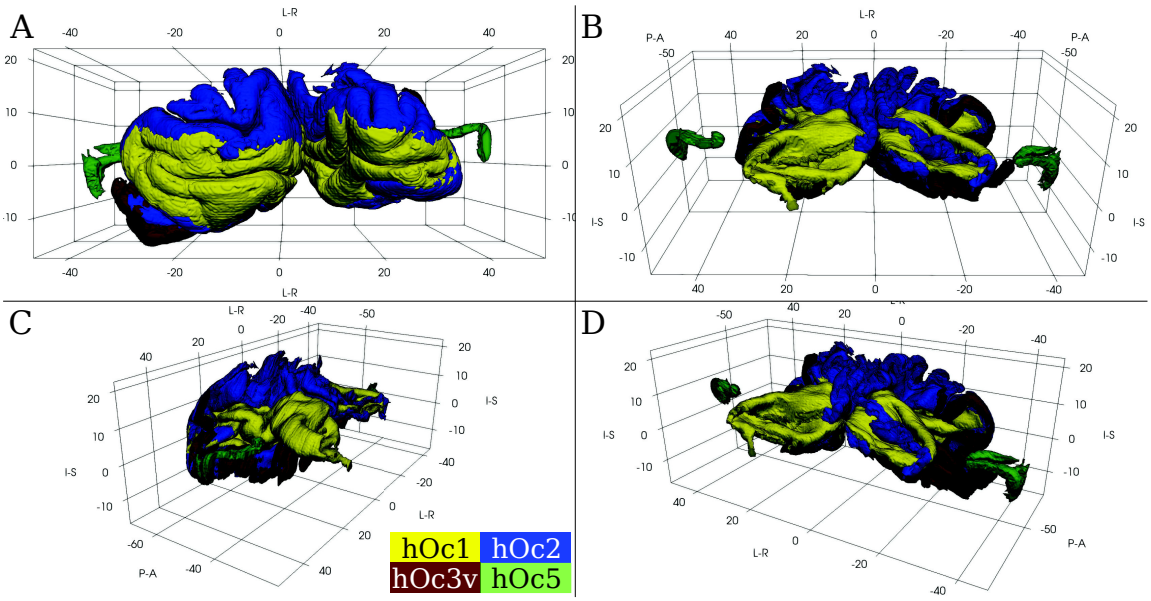



**Figure 5.12:** High-resolution 3D cytoarchitectonic maps of areas hOc1 (yellow), hOc2 (blue), hOc3v (red) and hOc5 (green) in the BigBrain (B20). The maps are created by using LSM models to automatically segment the brain areas on all relevant sections and assembling them into a consistent volume using the BigBrain reconstruction workflow (Section 3.2). The coordinate system specifies the location in the BigBrain space in mm. Letters indicate anatomical directions (L/R: left/right, P/A: posterior/anterior, I/S: inferior/superior). Visualization created using *ParaView* (Ahrens et al., 2005).

the BigBrain model (Section 2.1.3) to assemble high-resolution 3D maps from the segmentations. In the following, we describe the workflow used to create the maps.

### Quality control & generation of 3D cytoarchitectonic maps

We use the LSM approach with MS U-Net architecture to train a series of models for each of the four considered brain areas. We apply them to create segmentations for all sections without annotations. For areas hOc3v and hOc5, we use models trained with reduced distance between training sections (hOc3v* and hOc5*, Figure 5.6). In total, we train 18 models for areas hOc1 and hOc2, 23 models for hOc3v, and 7 models for hOc5. Areas hOc1 and hOc2 each cover 2461 sections, hOc3v covers 1441 sections, and hOc5 covers 541 sections.

The quality of the resulting segmentation is checked to validate the consistency and plausibility of the results and sort out sections with insufficient quality. In addition, sections containing histological artifacts (e.g., resulting from long-term storage or damaged tissue) are excluded from further processing. Between 8% (hOc3v) and

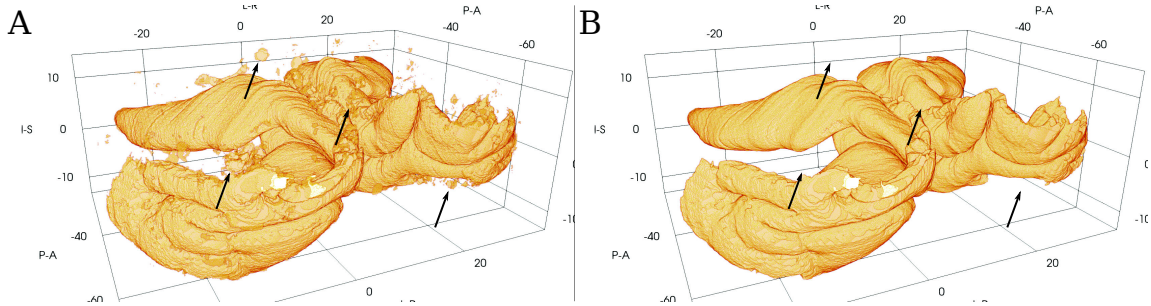



**Figure 5.13:** Illustration of the cleaning step applied to reconstructed cytoarchitectonic maps by the example of `hOc1`. The removal of small connected components suppresses small errors resulting from registration or prediction errors. Arrows mark examples of small errors that are removed by the cleaning step. The coordinate system specifies the location in the BigBrain space in mm. Letters indicate anatomical directions (L/R: left/right, P/A: posterior/anterior, I/S: inferior/superior). Visualization created using *ParaView* (Ahrens et al., 2005).

23% (`hOc1`) are excluded. Depending on the size of the respective area, the quality control step takes between several hours and a few days.

All sections that pass the quality check step are transformed into the reconstructed space of BigBrain (Section 3.2). Gaps in the series of transformed sections resulting from sections sorted out during quality control are filled by interpolating between available neighboring sections. Interpolation is performed based on Laplacian fields (Schober et al., 2016). As a result of the interpolation step, we obtain an uninterrupted series of segmentations, which are assembled into a 3D volume with 20 µm voxel resolution.

The obtained 3D volumes of each area are smoothed using a 3D median filter with a size of 11 × 11 × 11 voxels. The size of the filter is chosen based on the expected accuracy of annotation boundaries, which is estimated to be 100 µm (i.e., 5 voxels in each direction at 20 µm voxel resolution). After smoothing, we perform a connected component analysis and remove all components smaller than 27 $mm^3$ (3 mm × 3 mm × 3 mm). This step removes small errors resulting from registration or prediction errors (Figure 5.13). The resulting volume files have an uncompressed size of 85 GB (`hOc1` and `hOc2`), 50 GB (`hOc3v`), and 18 GB (`hOc5`). Preview files with a reduced resolution of 100 µm per voxels are created by downscaling to enable easier visualization. In a final step, the marching cubes algorithm (Lewiner et al., 2003) is used to create surface meshes of all areas at full and preview resolution.

The location, orientation, and morphology of the reconstructed cytoarchitectonic maps shown in Figure 5.12 are anatomically plausible and consistent. Their appearance aligns well with reference descriptions of the areas `hOc1`, `hOc2` (Amunts et al., 2000), `hOc3v` (Rottschy et al., 2007), and `hOc5v` (Malikovic et al., 2016).

**Table 5.2:** Volumes (in $mm^3$) of 3D cytoarchitectonic maps of different areas in B20. The reference mean $\mu$ and standard deviation $\sigma$ are computed based on male subjects from Amunts et al. (2007b). Volumes are corrected for tissue shrinkage (Section 3.2) using a shrinkage correction factor of 1.931.

| area | volume | corrected | $\mu$ | $\sigma$ | z-score |
|---|---|---|---|---|---|
| hOc1 | 9019.30 | 17416.27 | 18042.20 | 2464.39 | -0.25 |
| hOc2 | 6448.60 | 12452.26 | 12634.20 | 2862.84 | -0.06 |
| hOc3v | 1974.76 | 3813.26 | n.a. | n.a. | n.a. |
| hOc5 | 304.10 | 587.21 | 1144.40 | 406.53 | -1.37 |

### Volume and surface area computation

We compute the volume and the surface area of the created cytoarchitectonic maps and compare them to reference values from the literature. This step allows us to assess the anatomical plausibility of the created 3D cytoarchitectonic maps and the underlying segmentation results of the LSM method.

We compute the volume by multiplying the number of voxels in the created maps by the physical volume of each voxel ($20\,\mu m \times 20\,\mu m \times 20\,\mu m$). The computation of the surface area requires a more sophisticated approach. For cytoarchitectonic areas, the surface area typically refers to the area *at the brain surface*. This area is smaller than the total surface area of the created meshes. Thus, we need to identify those vertices and triangles of the surface mesh that correspond to the pial surface (the "outside") to obtain comparable estimates of the surface area.

To determine the subset of triangles on the pial surface, we first estimate the cortical depth within the cortex. The cortical depth specifies the distance of a voxel within the cortex to the pial boundary. We estimate the cortical depth by applying the procedure described in Leprince et al. (2015) to the cortical ribbon of the BigBrain model (Lewis et al., 2014, Section 3.2). The result is a volumetric dataset that specifies the cortical depth as a value between 0 and 1 for voxels in the cortex. We use the cortical depth dataset to assign the respective cortical depth to each vertex in the created surface meshes. Finally, we obtain the surface area of each cytoarchitectonic map by summing up the area of all triangles with a cortical depth below 0.25. The cortical depth of a triangle is determined by the maximum cortical depth across its associated vertices.

We report the estimated volumes and surface areas of the created cytoarchitectonic maps in Table 5.2 and Table 5.3, respectively. Volumes and surface areas are corrected to account for shrinkage of the brain tissue during histological processing (Section 3.2). To evaluate the anatomical plausibility of the created cytoarchitectonic maps, we compare volume and surface areas to values reported in Amunts et al. (2007b), who report shrinkage corrected volumes and surface areas for areas hOc1,

**Table 5.3:** Surface areas (in $mm^2$) of 3D cytoarchitectonic maps of different areas in B20. The reference mean $\mu$ and standard deviation $\sigma$ are computed based on male subjects from Amunts et al. (2007b). Surface areas are corrected for tissue shrinkage (Section 3.2) using a shrinkage correction factor of 1.551 ($1.931^{2/3}$).

| area | surface | corrected | $\mu$ | $\sigma$ | z-score |
|---|---|---|---|---|---|
| hOc1 | 6891.03 | 10685.76 | 12213.00 | 2225.55 | -0.69 |
| hOc2 | 6749.64 | 10466.52 | 10390.40 | 2925.37 | 0.03 |
| hOc3v | 2142.04 | 3321.62 | n.a. | n.a. | n.a. |
| hOc5 | 319.79 | 495.89 | 450.20 | 135.92 | 0.34 |

hOc2, and hOc5 in ten postmortem brains (five male, five female). To ensure a fair comparison, we compute mean and standard deviation across the five male subjects reported in this study, as BigBrain was created from the brain of a male donor.

The estimated volumes and surface areas correspond well with the values reported in Amunts et al. (2007b), indicating a high anatomical plausibility of the created cytoarchitectonic maps. The surface areas for hOc1, hOc2, and hOc5, as well as the volumes for hOc1 and hOc2 are largely confirmed with the reference values, while the volume for hOc5 is smaller compared to the reference values.

### 5.2.7 Web application for interactive brain mapping

The method is designed to support neuroanatomists in cytoarchitectonic mapping of large section series and thus make segmentation of complete cytoarchitectonic areas practically feasible. However, applying the method involves several non-trivial technical steps, including formatting and preparation of annotations, execution and monitoring of jobs HPC jobs, and visualization of prediction results. Thus, it is crucial to provide a user interface that enables users to annotate brain areas, define training sections, start and monitor the training process, create predictions using trained models, and investigate the prediction results.

We developed the web application ATLaSUI (Automatic Tissue Labeling System - User Interface)[2], which provides a user friendly interface to the ATLaS framework. The application is implemented as an extension of the software Microdraw[3], a web-based viewer and annotation tool for high-resolution image data, which enables the annotation of structures on large images. ATLaSUI makes use of the visualization and annotation capabilities of Microdraw and extends it by the possibility to apply LSMs.

[2] https://jugit.fz-juelich.de/c.schiffer/atlasui
[3] https://github.com/neuroanatomy/microdraw

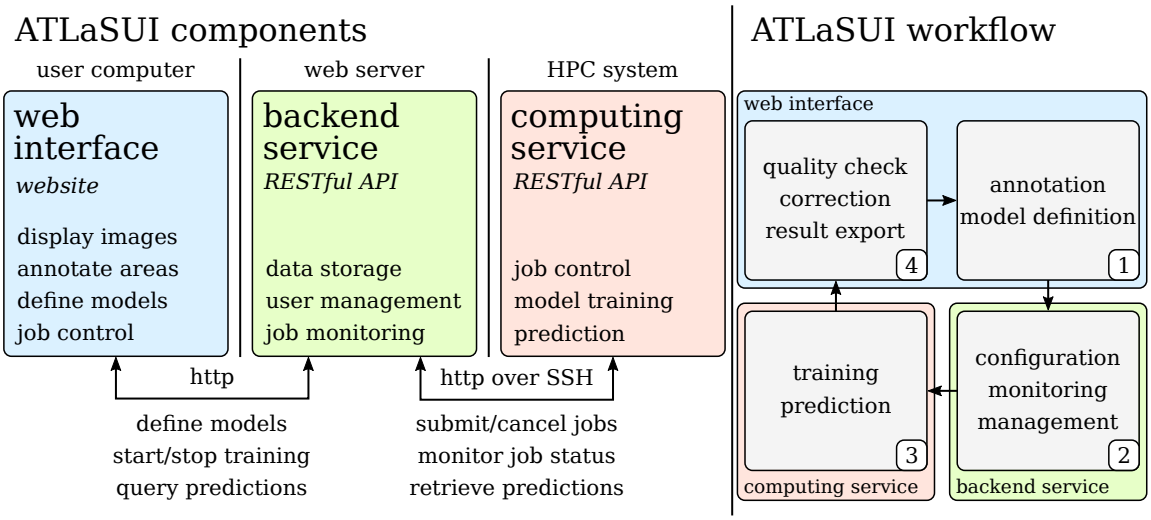



**Figure 5.14: Left**: Components of the ATLaSUI application for interactive brain mapping with LSMs. The web interface allows users to view images, create annotations, define models to train, and start the training process. The backend service stores data used by the web interface and coordinates communication with the computing service, which performs training and prediction on a HPC system. After training, the backend service can retrieve created predictions from the computing service to display them in the web interface. **Right**: Typical user workflow for ATLaSUI. Users annotate brain areas and define models using the web interface. The backend service configures training jobs and submits training tasks by communicating with the computing service. After training and prediction are finished, users can check, correct, or export the resulting predictions using the web interface.

ATLaSUI consists of three main components (Figure 5.14, left): A web interface, a backend service, and a computing service.

The web interface is a graphical user interface that is embedded into the existing user interface of Microdraw (Figure 5.15). It provides functions to specify which annotations should be used for model training, start the training and prediction processes, and inspect the prediction results obtained by trained models. In addition, it provides import functionalities for interoperability with other annotation tools and enables users to organize multiple models (e.g., multiple LSMs) in projects. The user interface is implemented using the frontend frameworks *Vue.js* and *Bootstrap*.

The *backend service* provides a *RESTful* application programming interface (API) that implements the backend logic of the web interface. It is implemented using the Python web framework *flask* and uses a document-oriented *MongoDB* database to store data. The API provides endpoints to store and retrieve information on annotations and projects, forwards requests to the computing service, and provides functions to query and visualize created predictions.

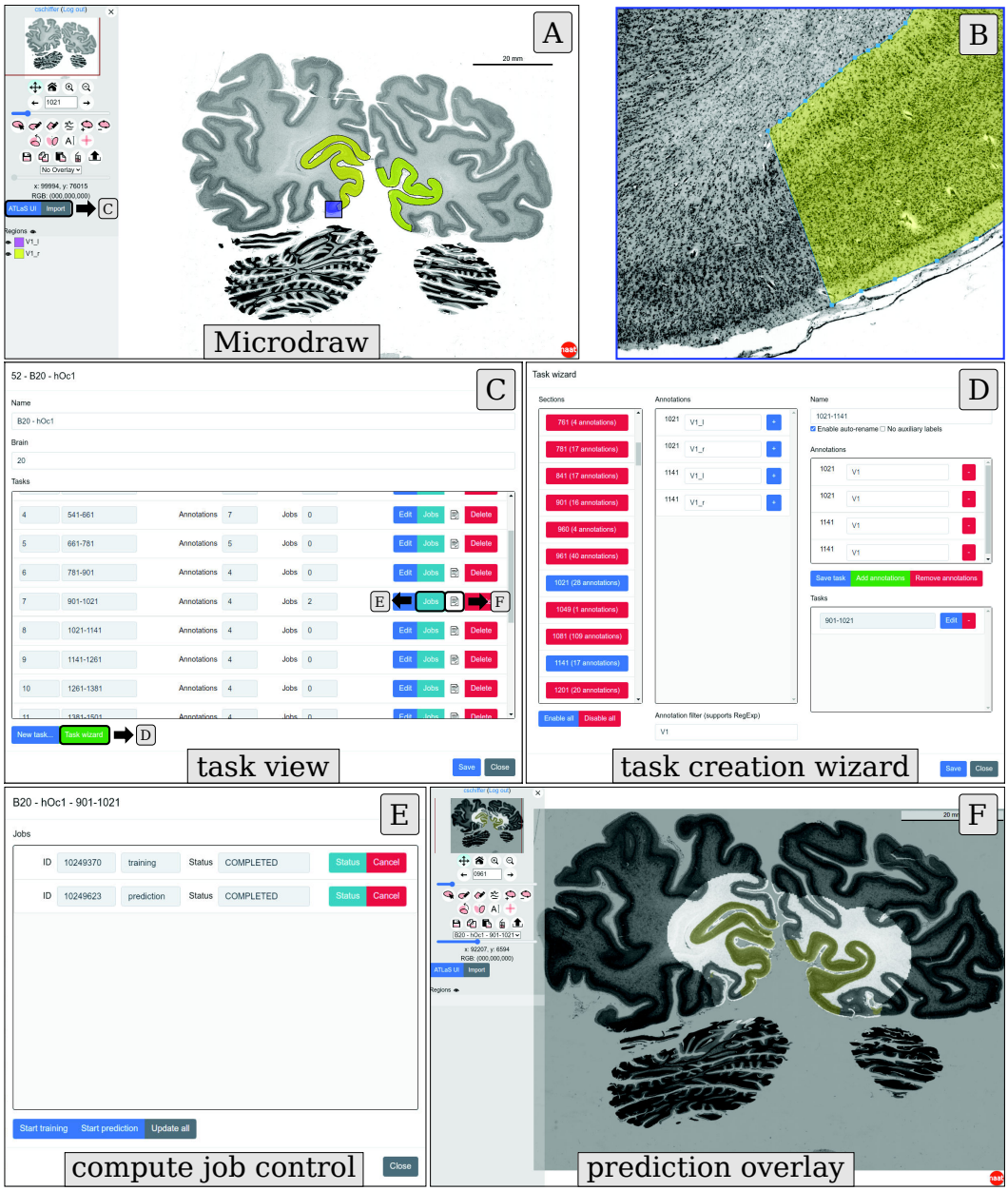



**Figure 5.15:** ATLaSUI web application for cytoarchitectonic mapping supported by deep learning. **A**: Microdraw allows displaying high-resolution images (here section 1021, `B20`) and annotating cytoarchitectonic areas (here `hOc1`, yellow). The blue square marks the location of the closeup view (B). **C**: Users define training tasks based on created annotations. One tasks corresponds to one deep neural network training. Tasks are created using a wizard (D). **E**: HPC compute jobs for training and prediction can be started and monitored from the interface. For the shown example (`hOc1`, sections 901 to 1021), training and prediction for 121 sections took 12 min and 15 min, respectively. **F**: Prediction results (here for section 961) can be inspected as semi-transparent overlay in Microdraw.

The computing service is a *RESTful* service that runs on a HPC system (e.g., JURECA-DC at JSC). Like the backend service, the computing service is implemented as a *flask* web service. It provides an API to submit, cancel, and query the status of compute jobs on the HPC system. When a user initiates the training of a model through the web interface, a request is sent to the backend service. The backend service prepares a compute job definition and submits it to the computing service. The same workflow is used to query the status of jobs or cancel running jobs, enabling full control over training and prediction from the user interface. As the HPC system running the computing service is typically not accessible through web protocols (e.g., *http*), the machines running the backend and HPC services are securely connected through a secure shell (SSH) tunnel.

ATLaSUI facilitates an interactive workflow for cytoarchitectonic brain mapping supported by deep learning, which integrates well with the existing mapping workflow (Section 3.3). As soon as a user has mapped a specific cytoarchitectonic area on two sections (Figure 5.15, A), ATLaSUI allows to define a training task (Figure 5.15, C, D) based on the created annotations and initiate the training process (Figure 5.15, E). ATLaSUI schedules and monitors the training job running on the underlying HPC system while the user continues annotating the next sections. When training finishes, predictions are automatically created for sections in-between the annotated training sections, which can be inspected through ATLaSUI (Figure 5.15, F). The user may then decide to add additional annotations to improve the segmentation quality (Section 5.2.3) or to continue working on the next section interval or brain area.

## 5.3 Summary

This chapter introduced a deep learning method for mapping of individual cytoarchitectonic areas in large series of brain sections. The developed LSM models trade generalizability for accuracy by specializing in local brain regions and specific cytoarchitectonic areas. Using a specifically designed training procedure and model architecture, LSMs achieve high segmentation performance and make mapping of complete cytoarchitectonic areas across large sections series feasible for the first time.

The method performs well across different areas (Section 5.2.1) and brain samples (Section 5.2.2). It can be flexibly adapted to the characteristics of different areas (Section 5.2.3), as it allows to adaptively provide additional annotated data that is efficiently used to improve segmentation performance. This property makes the method particularly well suited for interactive use. Interactive use is facilitated by the implemented web application ATLaSUI (Section 5.2.7).

We demonstrate the practical value of the method by creating high-resolution 3D maps of four cytoarchitectonic areas in the BigBrain dataset (Section 5.2.6). The reconstructed cytoarchitectonic maps represent the first high-resolution cytoarchitectonic maps created from a full series of individual histological brain sections. They enable the study of area-specific morphological and cytoarchitectonic features of the brain with unprecedented detail. The maps are released as part of the multilevel human brain atlas on EBRAINS[4] (Schiffer et al., 2019a,b, 2020a,b).

Section 8.1 provides a comprehensive discussion of the presented method and results.

[4] www.ebrains.eu

# 6 Large-scale cytoarchitectonic mapping with contrastive learning

Spitzer et al. (2017, 2018b) (Section 2.2.7) demonstrated the feasibility and potential of deep learning for classifying several cytoarchitectonic areas from the visual system. We extend this existing work to a larger scale by systematically investigating the applicability of deep learning methods for classifying many areas from different brain samples. Training deep neural networks for accurate classification of many cytoarchitectonic areas represents an important next step towards automated cytoarchitecture analysis at large scale.

We introduce a contrastive learning (Hadsell et al., 2006; Chen et al., 2020; Khosla et al., 2020) method that learns visual representations (Section 2.2.5) for automated cytoarchitecture classification from a large dataset of annotated cytoarchitectonic areas. The method learns to capture the subtle differences between brain areas through pairwise comparison of high-resolution image patches extracted from different cytoarchitectonic areas. Chen et al. (2020) and Khosla et al. (2020) demonstrated that self-supervised (Section 2.2.5) and supervised (Section 2.2.5) contrastive learning methods are able to learn good visual representations from images, enabling accurate image classification.

We first establish a baseline (Section 6.2.1) by training a model with categorical cross-entropy loss, and compare the performance of the baseline to models trained with contrastive learning. We examine the performance of the contrastive learning method when using different model architectures (Section 6.2.2), investigate its transferability to unseen brain samples (Section 6.2.3) and brain areas (Section 6.2.4), and its dependence on annotated training samples (Section 6.2.5). Finally, we study robustness to common data variations (Section 6.2.6), conduct an exploratory analysis of learned feature representations (Section 6.2.7), and investigate the limitations of self-supervised contrastive learning methods (Chen et al., 2020) for the given task (Section 6.2.8).

The method reported in this chapter was published in the article "Contrastive Representation Learning For Whole Brain Cytoarchitectonic Mapping In Histological Human Brain Sections" (Schiffer et al., 2021a). This chapter extends the published results by more model architectures, an analysis of the method's behavior in different

application scenarios, as well as an in-depth investigation of the learned feature representations.

## 6.1 Methods

### 6.1.1 Supervised contrastive learning for brain mapping

We use contrastive learning (Section 2.2.5) to learn visual representations for automated cytoarchitecture classification. We use a variant of the supervised contrastive loss proposed by Khosla et al. (2020) described in Section 2.2.5. Khosla et al. (2020) perform contrastive learning by regarding two data samples as similar if they are views (i.e., differently augmented versions) of the same source image *or* if they belong to the same class. In contrast to this, we do *not* use data augmentation to create multiple views of images in each batch and thereby do not define similarity based on data augmentation. Instead, we sample a batch of $b$ image patches $\mathbb{B} = \{X_1, \cdots, X_b\}$ and regard two samples as similar if and only if they share the same class label. We discard the data augmentation-based similarity relationship because we found that typical data augmentation operations are not sufficient to learn good visual features for cytoarchitecture classification. A discussion of this behavior and accompanying experimental studies are given in Section 6.2.8.

We adapt the supervised contrastive loss by Khosla et al. (2020) (Equation 2.59) and propose the following formulation:

$$L^{\text{sup}}(\mathbb{B}) = \frac{1}{b} \sum_{i=1}^{b} l^{\text{sup}}(i) \tag{6.66}$$

$$\text{with } l^{\text{sup}}(i) = -\frac{1}{n_{c_i}} \sum_{j=1}^{b} \mathbb{I}_{i \neq j} \mathbb{I}_{c_i = c_j} \log \frac{\exp\big(\text{sim}\big(\boldsymbol{z}_i, \boldsymbol{z}_j\big) / \tau\big)}{\sum_{k=1}^{b} \mathbb{I}_{i \neq k} \exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k) / \tau)}, \tag{6.67}$$

where $c_i$ is the class of the image with index $i$ and $n_{c_i}$ is the number of samples in the batch belonging to $c_i$. An encoder

$$f_E(X_i) = \boldsymbol{h}_i \in \mathbb{R}^{d_h} \tag{6.68}$$

maps an image $X_i$ to a $d_h$-dimensional feature space. A projection head

$$f_P(\boldsymbol{h}_i) = \boldsymbol{z}_i \in \mathbb{R}^{d_z} \tag{6.69}$$

then maps the feature vector $\boldsymbol{h}_i \in \mathbb{R}^{d_h}$ to a $d_z$-dimensional projection space, where the contrastive loss is computed.

The projection head is discarded after the contrastive pre-training stage. Following the linear evaluation protocol, a linear classifier $f_C(\boldsymbol{h}_i) = \hat{\boldsymbol{y}}_i$ is then trained using the learned features (Figure 2.16).

For this method, we model cytoarchitectonic brain mapping as a *classification task*. Given an image patch extracted from the approximate center of the cortex, the model is tasked to predict the cytoarchitectonic area from which the patch was extracted. In comparison, Spitzer et al. (2017, 2018b) and the LSMs in Chapter 5 model the task as a *segmentation task* (i.e., each *pixel* is assigned to the respective cytoarchitectonic area it belongs). Using a classification approach instead enables us to use contrastive learning methods without the need to adapt them for segmentation problems. Image patches with a typical size (i.e., $2 \times 2\,\mathrm{mm}^2$ to $4 \times 4\,\mathrm{mm}^2$) typically show only a single cytoarchitectonic area (i.e., all cortex pixels in the image belong to the same class). As such, the additional technical and methodological efforts associated with segmentation methods (e.g., memory demands, decoder design) are not justified for the given task. If required, segmentation masks can be derived by combining pointwise predictions with a cortex segmentation (e.g., by "coloring" each cortex pixel based on the closest predicted point).

### 6.1.2 Dataset preparation

This section describes the creation of training and evaluation datasets from the microscopic image data described in Chapter 3. We consider a total of 113 cytoarchitectonic areas. A list of these areas (Table 3.2) and details on available annotations are given in Section 3.3. For the experiments presented in this chapter, we use data from eight brains $\mathcal{B} = \{\texttt{B01}, \texttt{B03}, \texttt{B04}, \texttt{B05}, \texttt{B06}, \texttt{B07}, \texttt{B10}, \texttt{B12}\}$ (Table 3.1). We decide to exclude B20 from these experiments, as the large number of sections in B20 requires specific processing steps. However, the presented methods are generally compatible with B20 or comparable datasets. Note that not all areas are mapped in every brain (Figure 3.6).

All but one brain are used to create training and test samples. Training samples are used to train the model. Test samples are used to evaluate the model's transferability to new samples from known brains[1]. The additional hold-out brain is used to examine the transferability of trained models to unknown brains (i.e., to brains from which no samples were seen during training). We refer to the hold-out brain $B_h$ as *transfer brain*. Section 6.2.3 presents experiments using different transfer brains. The set of brains used for training and testing is referred to as $\mathcal{B}_{tt} = \mathcal{B} \setminus \{B_h\}$ (training and testing).

[1] We use the terms *known* and *unknown* as synonyms for "seen during training" and "not seen during training", respectively.

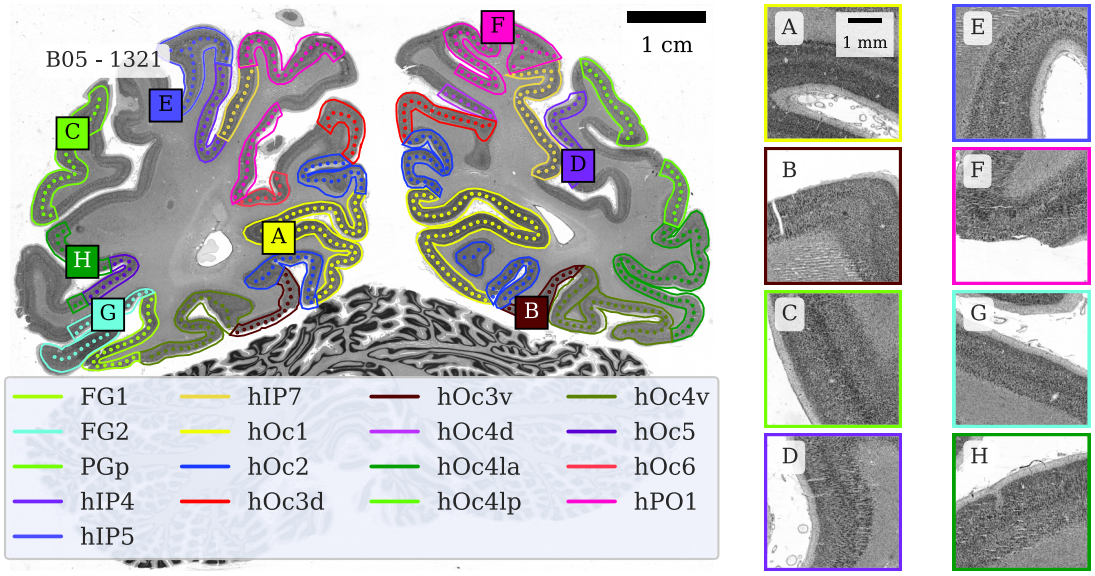



**Figure 6.1:** Image patches sampled from the cortex. Cytoarchitectonic areas are annotated as contours on the brain section. Center points (colored dots) for image patches are sampled uniformly along the midline of each annotation. Example patches on the right represent typical image patches with a size of approximately $4 \times 4\,\mathrm{mm}^2$.

Annotated sections belonging to brains in $\mathcal{B}_{tt}$ are denoted as $\mathcal{S}_{tt}$. They are split into *training sections* $\mathcal{S}_{tr} \subset \mathcal{S}_{tt}$ and *test sections* $\mathcal{S}_{te} = \mathcal{S}_{tt} \setminus \mathcal{S}_{tr}$. Sections are split with an 80-20 ratio, so a fifth of the annotated sections are used to estimate the performance on unknown sections. We sort sections according to their natural ordering (i.e., their section number) and select every fifth section as a test section to perform the split. This approach differs from the random splitting approach often used in machine learning applications. We use the fixed split procedure to ensure that test sections are distributed approximately uniform across brains and include samples from all considered areas. Annotated sections from the transfer brain $B_h$ are denoted as $\mathcal{S}_h$.

Image patches for training, testing, and transferability evaluation are extracted from sections $\mathcal{S}_{tr}$, $\mathcal{S}_{te}$, and $\mathcal{S}_h$, respectively, by uniformly sampling image patches along the midline of the cortex (Figure 6.1). The midline of an annotation is computed from its morphological skeleton. We uniformly sample points with a distance of 1 mm along the midline of each annotation. Datasets for training, testing, and transferability evaluation are created by extracting square image patches centered at each of the sample points.

Cytoarchitectonic areas vary in size, so the number of image patches per area varies as well. We counter the resulting class imbalance by resampling the training dataset (with replacement) such that it contains a constant number of image patches from

each area. Thus, some areas are undersampled (i.e., not all available image patches are included in the training set), and some areas are oversampled (i.e., some image patches are included multiple times in the training set).

We define the *sampling rate* $n_s$ as the number of image patches sampled per area $\mathtt{A} \in \mathcal{A}$. We then define the *sampling ratio* $r_\mathtt{A} = n_s \,/\, n_\mathtt{A}^t$ of an area $\mathtt{A}$ as the ratio of the sampling rate $n_s$ and the total number of available image patches for that area $n_\mathtt{A}^t$. Undersampled areas have a sampling ratio between 0 and 1, while oversampled areas have a sampling ratio larger than one. We determine the sampling rate $n_s$ such that the median sampling ratio $\bar{r}_\mathcal{A} = \text{median}_{\,\mathtt{A}\in\mathcal{A}}\, r_\mathtt{A}$ across all considered cytoarchitectonic areas is close to one to ensure that the number of samples is balanced between areas. Using this strategy, we select a sampling rate of $n_s = 1200$ image patches per area. Therefore, the training dataset consists of $113 \times 1200 = 135\,600$ image patches, where 113 is the number of cytoarchitectonic areas considered. Datasets for testing and transferability evaluation are not resampled.

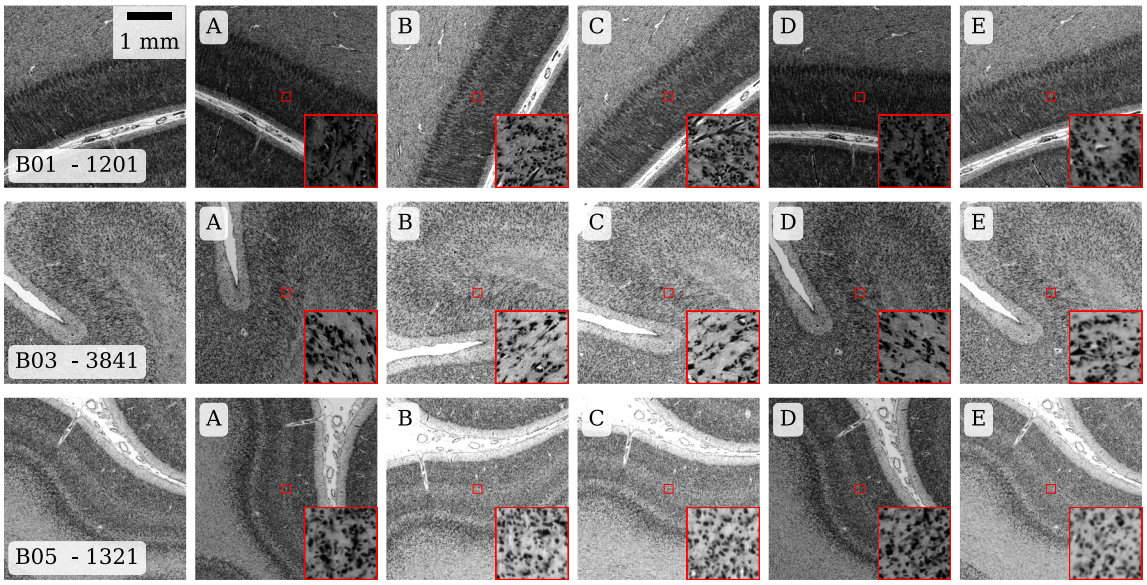
### 6.1.3 Data augmentation

We apply data augmentation to reflect natural variations in the data and artificially increase the amount of training data (Figure 6.2).

The center location of each image patch is randomly translated before it is extracted. The translation ensures that no image patch is extracted from the exact same location multiple times. In practice, it cannot be expected that image patches are extracted from the exact center of the cortex, so introducing small variations to the patch placement improves robustness. The center of each patch is independently translated into a random direction. The distance of the translation is sampled from a uniform distribution $d \sim U[0, 200\,\mu\text{m}]$.

Each image patch is rotated by a random angle $\theta \in U[-\pi, +\pi]$ and randomly mirrored with a probability of 50%. These geometrical transformations account for the fact that image patches can be rotated arbitrarily, mainly due to the convoluted morphology of the cortex and the arbitrary orientation of histological sections on the microscopy slides. Artificially applying these transformations during training improves the robustness of the model and enables it to classify image patches with arbitrary orientations.

Random intensity augmentation operations are applied to each image patch. They reflect naturally observed intensity variations originating from the histological preparation (e.g., staining variations) and inter-individual differences. The intensities $x \in [0, 1]$ of all pixels in an image are augmented according to $\alpha x^\gamma + \beta$. The parameters $\alpha \sim U[0.9, 1.1]$, $\beta \in U[-0.1, +0.1]$ and $\gamma = \frac{\log\left(0.5+2^{-0.5}Z\right)}{\log\left(0.5-2^{-0.5}Z\right)}$ $(Z \sim U[-0.05, +0.05])$ are randomly sampled for each image patch and then applied for all pixels of a patch.




| | A | B | C | D | E |
|---|---|---|---|---|---|
| rotation $\theta$ | $\frac{\pi}{4}$ | $-\frac{\pi}{4}$ | $-\frac{\pi}{8}$ | $\frac{\pi}{8}$ | - |
| translation $d$ | 200 µm | −200 µm | 141 µm | 141 µm | - |
| contrast $\alpha$ | 0.9 | 1.1 | 1.05 | 0.95 | 1.02 |
| brightness $\beta$ | -0.05 | 0.05 | 0.10 | -0.10 | - |
| gamma $Z$ | - | -0.01 | -0.02 | 0.01 | 0.05 |
| blurring $\sigma$ | - | - | - | 0.25 | 0.8 |
| sharpening $\sigma$ | - | - | 0.25 | - | - |

**Figure 6.2:** Example image patches with different data augmentation operations. The left column shows three original image patches with a size of approximately $4 \times 4\,\mathrm{mm}^2$. Columns A to E show different augmented versions of the original image patch in the respective row. Red rectangles show small image patches with higher magnification to illustrate the effect of data augmentation at the cellular level. Augmentation parameters are shown in the table.

The sampling procedure for the *gamma augmentation* ($x^\gamma$) ensures that the operation is unbiased (i.e., $\mathbb{E}\left[x^\gamma\right] = x$), as proposed by Pohlen et al. (2017).

We apply random Gaussian blurring and sharpening. Image patches are blurred with an isotropic Gaussian kernel $G_\sigma(x)$ (standard deviation $\sigma \sim U[0.125, 1.0]$), or sharpened according to $x + \delta(G_\sigma(x) - x)$ ($\sigma \sim U[0.125, 2.0]$, $\delta \sim U[0.5, 1.5]$) with probability 25%, respectively. Blurring and sharpening mimic sharpness variations introduced during image acquisition (e.g., focus variations of the microscopic scanners).

Random data augmentation ensures that the exact same image patch is unlikely to be encountered twice during training. This is important to consider given that the class balancing procedure described in Section 6.1.2 can result in multiple instances

of an image patch being included in the training dataset. Even if the same image patch location is included multiple times within the dataset (or even within a single training batch), the described data augmentation pipeline is likely to augment each patch instance in very different ways, making all instances potentially valuable for the training progress. Pre-training and finetuning use the same data augmentation pipeline.

### 6.1.4 Training parameters

This section details the default training parameters for the experiments described in Section 6.2. If not stated otherwise, these parameters are used for all presented experiments.

The contrastive pre-training stage comprises training for 150 epochs (i.e., 150 passes through all image patches in the training dataset). The linear classifier in the finetuning stage is trained for 30 epochs. The remaining training parameters are identical for pre-training and finetuning if not stated otherwise.

Following Chen et al. (2020) and Khosla et al. (2020), optimization is performed using the LARS optimizer (Section 2.2.1). The LARS optimizer is combined with Nesterov momentum (Section 2.2.1, Sutskever et al., 2013) with momentum $\mu = 0.9$ and weight decay $\omega = 0.0001$ for all non-bias parameters. The trust parameter of LARS is set to $\eta = 0.02$ (Equation 2.11). The temperature parameter for the supervised contrastive loss (Equation 6.67) is set to $\tau = 0.07$.

Contrastive learning methods rely on large batch sizes to provide sufficient negative pairs for the loss computation (Chen et al., 2020). The batch size is set to $b = 2048$ image patches per batch. Depending on the experiment, the learning rate is set to either $0.01\frac{b}{128} = 0.16$ or $0.005\frac{b}{128} = 0.08$. The learning rate is kept constant during training. Expressing the learning rate depending on the batch size is common practice to adapt it for larger or smaller batch sizes (e.g., due to memory constraints). If not stated otherwise, brains $\mathcal{B}_{tt} = \{\texttt{B01}, \texttt{B03}, \texttt{B04}, \texttt{B05}, \texttt{B06}, \texttt{B10}, \texttt{B12}\}$ are used for training and testing, while $B_h = \texttt{B07}$ is used as transfer brain.

Training is implemented using the ATLaS framework (Section 4.2). The large image patch size and the large batch size make distributed training across multiple compute nodes of a HPC system mandatory to meet memory requirements and enable training in practically feasible time frames (i.e., several hours). The contrastive pre-training uses 16 compute nodes of JURECA-DC. Finetuning uses 4 compute nodes of JURECA-DC. Table 6.1 gives an overview of the computational resources used for training.

Training is distributed using data-parallel DDL (Section 2.2.6). The feature vectors $\boldsymbol{h}$ are collected from all participating GPUs before computing the contrastive loss. This synchronization ensures that negative pairs from the entire batch are available

**Table 6.1:** Overview of computational resources used for pretraining and finetuning in the proposed contrastive learning workflow. Runtime estimates specify the average time it takes to train a model. Memory, CPU, and GPU count are computed as the product between node count and units per node. CPU and GPU hours are computed as the product between the runtime and the number of occupied CPUs and GPUs, respectively.

| | pre-training | finetune |
|---|---|---|
| # compute nodes | 16 | 4 |
| # CPU cores | 2048 | 512 |
| memory | 8192 GB | 2048 GB |
| # GPUs | 64 | 16 |
| GPU memory | 2560 GB | 640 GB |
| # epochs | 150 | 30 |
| data read (epoch) | 1059.4 GB | 1059.4 GB |
| data read (total) | 155.2 TB | 31.0 TB |
| training time | 4 h | 4 h |
| CPU hours | 8192 h | 2048 h |
| GPU hours | 256 h | 64 h |

for the loss computation, even if the feature vectors are computed on different GPUs. Batch normalization normalization statistics (Equations 2.27 and 2.28) are computed across all GPUs.

Training is performed using automatic mixed precision (AMP). AMP automatically identifies operations that can be safely performed using 16 Bit floating-point precision (*half precision*) without risking significant precision loss or incorrect results. It improves runtime performance and reduces memory requirements.

### 6.1.5 Network architectures

We investigate different architectures for the encoder $f_E$. The architectures are shown in Figure 6.3.

The `base` architecture is based on existing work for cytoarchitecture segmentation by Spitzer et al. (2017, 2018b). It resembles the encoder of the modified U-Net by Spitzer et al. (2017), which is equivalent to the HR U-Net (Figure 5.5) described in Section 5.1.5.

`ResNet18`, `ResNet50`, and `ResNet101` are based on the ResNet architecture (Section 2.2.3) proposed by He et al. (2016a). `DenseNet121` is based on the DenseNet architecture proposed by Huang et al. (2017). We account for the large image patch size used in our experiments by replacing the initial downsampling block (i.e., the first two convolutional layers and the pooling layer) of these architectures with the downsampling block of the `base` architecture. Table 6.2 lists the number of parameters for each of the used architectures.

The projection head $f_P$, which maps the output of the encoder $f_E$ to a lower-dimensional space where the contrastive loss is computed, is implemented by a MLP

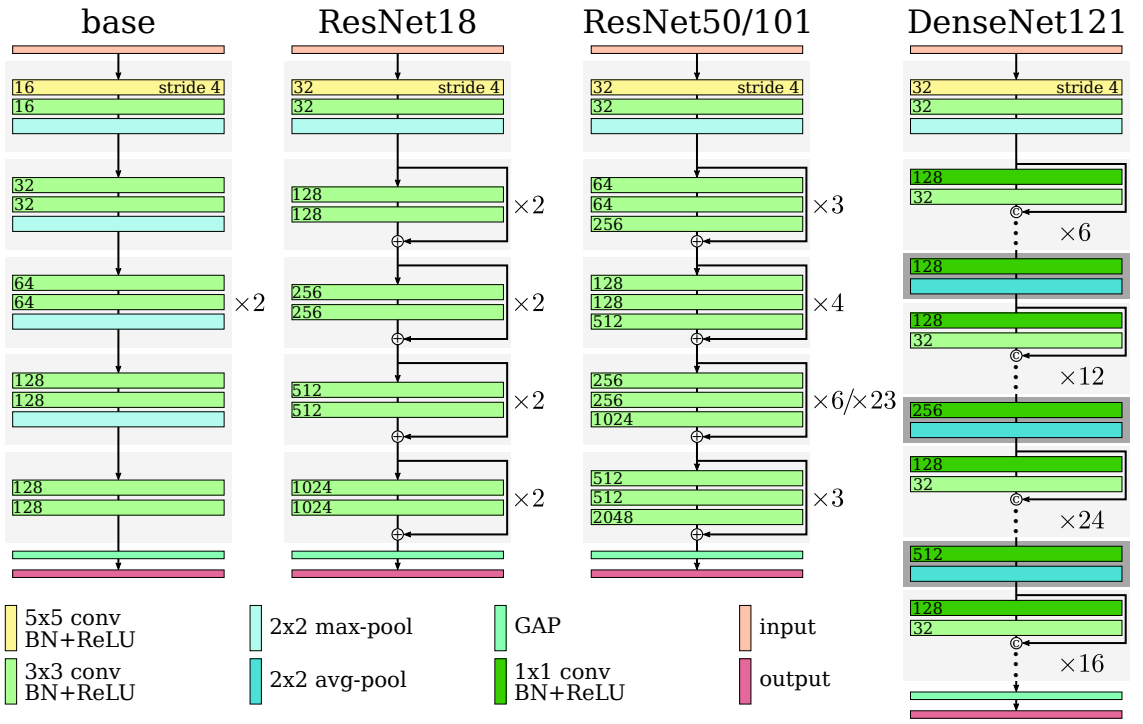



**Figure 6.3:** Model architectures used for contrastive learning. The `base` architecture closely follows the encoder of the modified U-Net architecture that Spitzer et al. (2017) proposed for cytoarchitecture segmentation. `ResNet18`, `ResNet50`, `ResNet101` (He et al., 2016a), and `DenseNet121` (Huang et al., 2017) are based on common architectures for image classification, with a modified downsampling block to account for the large image patch size. Numbers in the blocks denote the number of filters in the respective convolutional layer. The encircled "+" and "c" denote addition and concatenation along the feature dimension, respectively.

with two fully-connected layers and output dimensionality $d_z = 256$. The first fully-connected layer uses ReLU activation and batch normalization.

The total available GPU memory (2560 GB) is not sufficient to train the large `ResNet101` and `DenseNet121` architectures. We therefore use gradient checkpointing in combination with DDL to train these models (Section 2.2.6).

## 6.2 Results

We conduct experiments to study the benefit of contrastive learning for cytoarchitecture classification. We use the macro F1-score (i.e., the average of F1-scores per area, Equation 2.19) to quantify the performance of trained models. F1-scores are computed on image patches that are not seen during training. Image patches are ex-

**Table 6.2:** Number of model parameters and output dimensionality $d_h$ of the used architectures.

| model | # parameters | $d_h$ |
|---|---|---|
| `base` | 0.66M | 128 |
| `ResNet18` | 11.16M | 512 |
| `ResNet50` | 23.50M | 2048 |
| `ResNet101` | 42.49M | 2048 |
| `DenseNet121` | 6.85M | 1020 |

tracted from test sections $\mathcal{S}_{te}$ in known brains $\mathcal{B}_{tt}$ or from sections $\mathcal{S}_h$ in the transfer brain $B_h$. We refer to the respective scores as *test scores* and *transfer scores*.

In Section 6.2.1, we establish a baseline model trained with categorical cross-entropy as comparison for the contrastive learning method. The baseline also allows investigating aspects of the method that are not specific to contrastive learning (e.g., the input image size).

The subsequent experiments aim to answer the following questions:

- How does contrastive learning affect performance in comparison to the baseline model and using different model architectures? (Sections 6.2.1 and 6.2.2)
- How does a model trained with contrastive learning method transfer to unseen brains? (Section 6.2.3)
- How well do learned features transfer to unseen brain areas? (Section 6.2.4)
- How does the number of training samples affect classification performance? (Section 6.2.5)
- How robust are learned features under naturally occurring data variations? (Section 6.2.6)
- What kind of features does the model learn? (Section 6.2.7)

We also investigate the limitations of self-supervised contrastive learning (i.e., SimCLR) for cytoarchitecture classification in Section 6.2.8.

### 6.2.1 Baseline experiments with categorical cross-entropy loss

In this section, we establish a baseline model. If not stated otherwise, the training parameters detailed in Section 6.1.4 are used. Training is performed for 180 epochs (i.e., the sum of pre-training and finetuning epochs used during contrastive training) using categorical cross-entropy loss. We train models using the `base` architecture and the `ResNet50` architecture. In addition, we evaluate how pre-initialization of the `base` model with parameters pre-trained using the SSL method by Spitzer et al. (2018b) affects performance for the given task (Section 2.2.7).

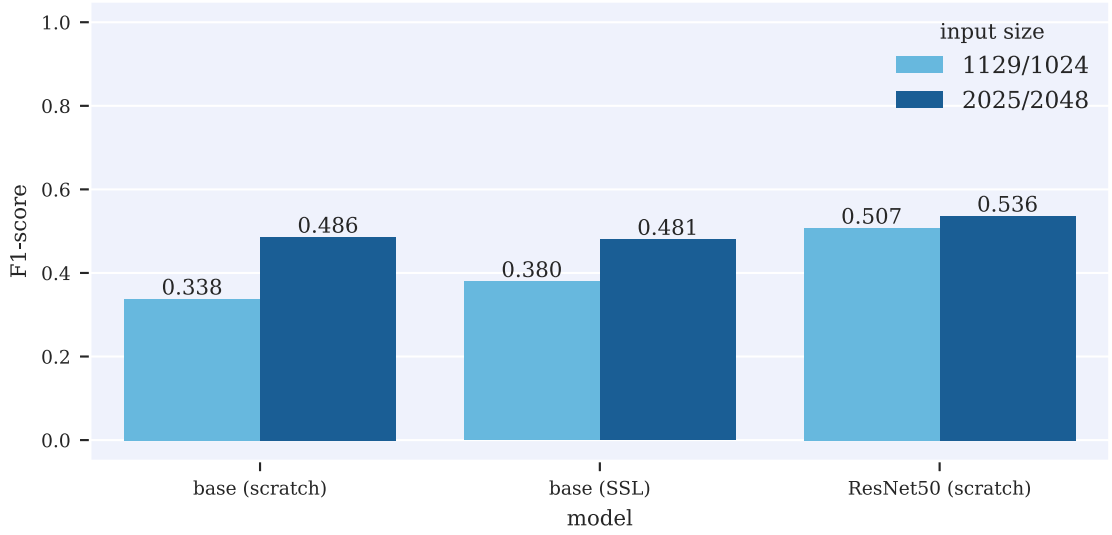



**Figure 6.4:** Test F1-scores of baseline models trained with categorical cross-entropy. Results are reported for the `base` architecture trained from scratch and pre-initialized from the SSL task by Spitzer et al. (2018b), and for the `ResNet50` architecture. For each architecture, scores obtained with image patch size of approximately 1000 px and 2000 px are reported.

We train each model with two different input patch sizes, which allows assessing how the input size affects performance. Patch sizes are set to 1129 px or 2025 px for the `base` architecture and to 1024 px or 2048 px for the `ResNet50` architecture. Image patches have a resolution of 2 µm/px, resulting in an approximate physical patch size of 2 mm or 4 mm. We train each model with learning rates $0.01\frac{b}{128} = 0.16$ and $0.005\frac{b}{128} = 0.08$, and report scores of the respective best performing model[2].

Scores obtained on the test sections ($\mathcal{S}_{te}$) are reported in Figure 6.4. Models trained with smaller image patches generally obtain lower scores. A plausible explanation for this observation is that an image patch size of approximately 1000 px ($\sim$ 2 mm) is not always sufficient to cover the entire cortex, which has a thickness of 2 mm to 4 mm (Von Economo, 1925; Zilles et al., 2012).

There is no considerable performance difference between the `base` model trained from scratch and the one pre-initialized from the SSL task (Spitzer et al., 2018b). The pre-initialized model performs slightly better when using smaller image patches, but the performance with larger image patches is almost identical. Pre-training using the SSL task proposed by Spitzer et al. (2018b) thus seems not to improve performance for the considered large-scale cytoarchitecture classification task.

[2]Learning rate 0.08 is used for `base` (scratch, 1129), `base` (SSL, 1129), `base` (SSL, 2025), and `ResNet50` (1024). Learning rate 0.16 is used for `base` (scratch, 2025) and `ResNet50` (2048).

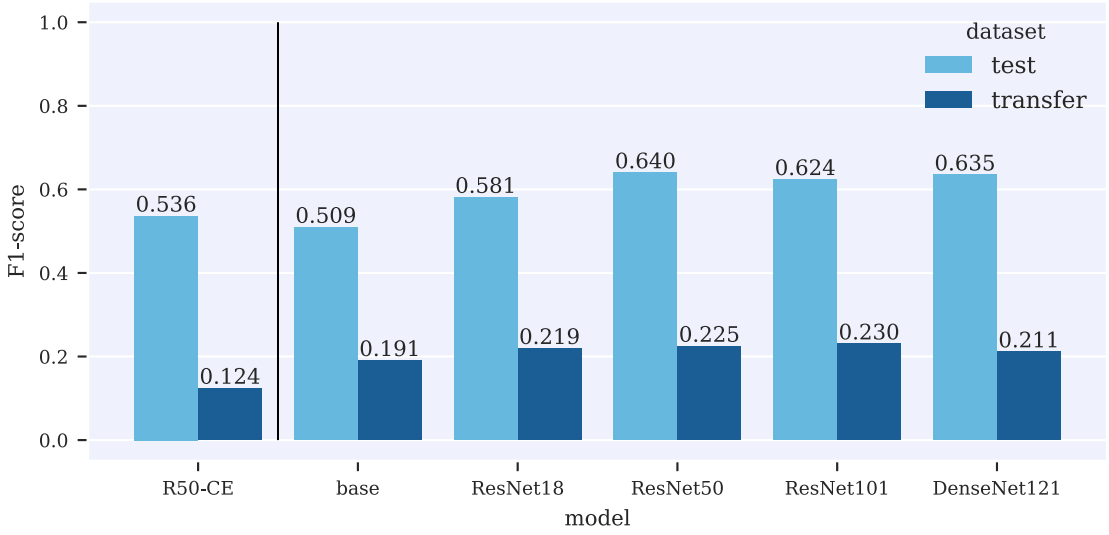



**Figure 6.5:** F1-scores obtained using models trained with supervised contrastive learning and different model architectures. Results are reported for test sections $\mathcal{S}_{te}$ (i.e., unknown sections from brain used during training), and for the transfer brain $B_h$ = B07. `R50-CE` denotes the best performing model trained with categorical cross-entropy (Figure 6.4). All models use an input image patch size of approximately 2000 px at 2 µm/px ($\sim$ 4 mm).

The `ResNet50` architecture outperforms the `base` architecture independently of the image patch size. The overall best performance is obtained by `ResNet50` with large image patches.

Based on these experiments, we choose the `ResNet50` model trained with categorical cross-entropy loss and an image patch size of 2048 px at 2 µm/px as a baseline for following experiments. We refer to the baseline model as `R50-CE` (`ResNet50` with categorical cross-entropy). As the results show that larger image patches improve performance, all following experiments use an image patch size of approximately 2000 px at 2 µm/px.

### 6.2.2 Architectures for supervised contrastive learning

We investigate the performance of the proposed contrastive learning method (Section 2.2.5). We assess the `base`, `ResNet18`, `ResNet50`, `ResNet101`, and `DenseNet121` architectures. The `base` architecture uses an input size of 2025 px, the other architectures use an input size of 2048 px. All image patches have a resolution of 2 µm/px.

We train each model with learning rates $0.01\frac{b}{128} = 0.16$ and $0.005\frac{b}{128} = 0.08$, and report scores of the best performing model[3].

Test and transfer F1-scores obtained with different architectures are reported in Figure 6.5. The baseline results (Section 6.2.1) are represented by `R50-CE`.

We first focus our analysis on the test scores. The baseline model `R50-CE` is outperformed by all models trained with contrastive learning, except for the `base` model. When both methods use the `ResNet50` architecture, the contrastive learning model achieves considerably higher test scores than the baseline model.

The `ResNet50` model obtains the highest test F1-scores. Despite its significantly higher number of parameters (Table 6.2) and higher representational capacity, the `ResNet101` model performs slightly worse than `ResNet50`. The performance increases with the model size only up to the size of `ResNet50`. This indicates that classification performance does not improve beyond a certain saturation point in terms of model size. It is interesting to note that the `DenseNet121` achieves scores comparable to those of `ResNet50` and `ResNet101`, although it has significantly fewer parameters.

The transfer F1-scores are considerably lower than the test scores, which shows that classification performance on unseen brains (here `B07`) is lower than on known brains. Training with contrastive learning leads to notably increased transfer scores compared to the baseline model. Transfer scores of the contrastive learning models are comparable, with only slight improvements from smaller to larger models.

Based on the results, we chose `ResNet50` trained with contrastive learning as the default model for the next experiments. We refer to this model as `R50-SCL` (ResNet50 with supervised contrastive learning). `DenseNet121` obtains comparable results with significantly fewer parameters than `ResNet50`, but its densely connected architecture makes it computationally expensive and thus less attractive from a practical perspective.

In the following sections, we investigate the properties of the method to gain a better understanding of its behavior in different application scenarios. In particular, we study the observed reduced classification performance on unseen brains in Section 6.2.3.

### 6.2.3 Classification performance on unseen brains

The results presented in Section 6.2.2 show a significant performance gap between test and transfer F1-scores. This raises the question whether the observed behavior is specific to the selected transfer brain (`B07`), or if the same effect also occurs for different transfer brains. We aim to answer this question by conducting experiments with different transfer brains.

[3] Learning rate 0.08 is used for `ResNet50`, `ResNet101`, and `DenseNet121`. Learning rate 0.16 is used for `base` and `ResNet18`

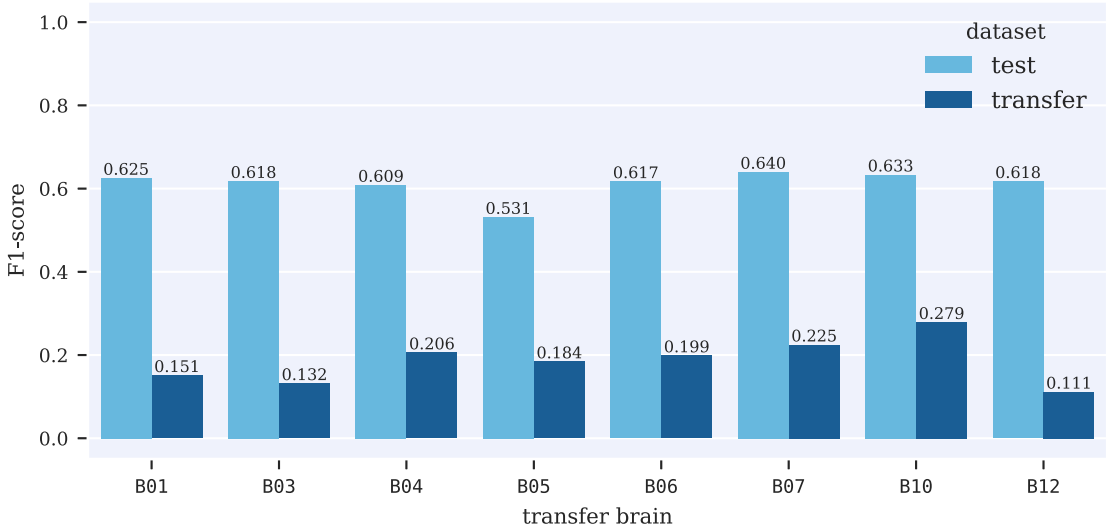



**Figure 6.6:** Test and transfer F1-scores obtained by models trained on different sets of training brains. Each experiment uses a different transfer brain $B_h$. The results show systematically decreased classification performance on the transfer brain compared to sections from known brains.

We conduct eight experiments, each using one of the eight considered brains (B01, B03-B07, B10, B12) as transfer brain $B_h$ and the respective remaining brains as training and test brains $\mathcal{B}_{tt}$. We conduct the experiments using the R50-SCL model (Section 6.2.2). Training, test, and transfer samples for each experiment are created using the workflow described in Section 6.1.2.

Figure 6.6 shows the test and transfer F1-scores for different choices of the transfer brain. The results show that the previously observed gap between test and transfer F1-scores is consistently observed independently of the choice of the transfer brain. All models perform worse on image patches sampled from an unseen brain. Note that each model is trained on different brains, so test and transfer F1-scores are not directly comparable among experiments.

The test scores obtained by different models are at a comparable level, indicating that none of the brains is particularly "more difficult" to classify than others. Based on this observation, we can assume that a model should be able to classify samples from a brain if training data for that particular brain is available.

An interesting practical question is *how much* data is required to enable classification of a particular brain, i.e., how many annotations from an unseen brain are required to achieve satisfactory classification performance. To answer this question, we conduct experiments that include additional training samples from the transfer brain in the training process and observe the effect on the classification performance.

The number of training samples in a brain can be quantified by the number of annotated areas and by the number of samples per area (Section 6.1.2). However, computational requirements prohibit an extensive evaluation along both of these dimensions. For our experiments, we fix the number of samples per area and only vary the number of annotated areas in the transfer brains. This scenario is practically relevant, as areas are typically only partially annotated in each brain (Figure 3.6).

We conduct experiments with the default dataset, which considers B07 as transfer brain. We split the annotated sections of B07 into training and test sections with a ratio of 80% to 20%. The defined test sections are used to compute F1-scores in the transfer brain. We vary the number of annotated areas in B07 by including random subsets with 25%, 50%, and 75% of all areas in the training set. We repeat each experiment with three random seeds and report the mean performance across runs to account for the randomness of this step. For reference, we report the performance of a model trained using all available areas (100%) and of a model that does not use any data from B07 for training (0%). The latter scenario (i.e., using no training data from the transfer brain) represents our default training protocol.

We sample 240 training samples per area from B07. We compute this sample count from the average number of samples per area and brain in the default training dataset. The samples from B07 are added to the training dataset. This approach simulates the process of acquiring additional training data in an unseen brain.

Figure 6.7 shows the results. Test scores remain relatively stable. Transfer scores increase when the proportion of training data from the transfer brain increases. Test and transfer scores reach comparable values when training data from all (100%) of the considered areas is provided in the transfer brain.

The results suggest that providing training examples for a small subset of areas (e.g., less than 50% of the considered areas) from the transfer brain does not improve classification performance. Providing training examples from the transfer brain for more than 50% of the considered areas improves the classification performance on the transfer brain. However, the transfer scores only reach the level of the test scores when training data for *all* considered areas is provided in the transfer brain.

An extended discussion of the limited performance on unseen brains is given in Section 8.2.

### 6.2.4 Transferability of learned features for unseen areas

The supervised contrastive learning method uses annotations of brain areas (113 in our experiments) to learn features for cytoarchitecture classification. A practically relevant question is if the learned features can be used to classify *unseen areas*, i.e., areas that are not used (or not yet known) when the feature extraction model is trained.

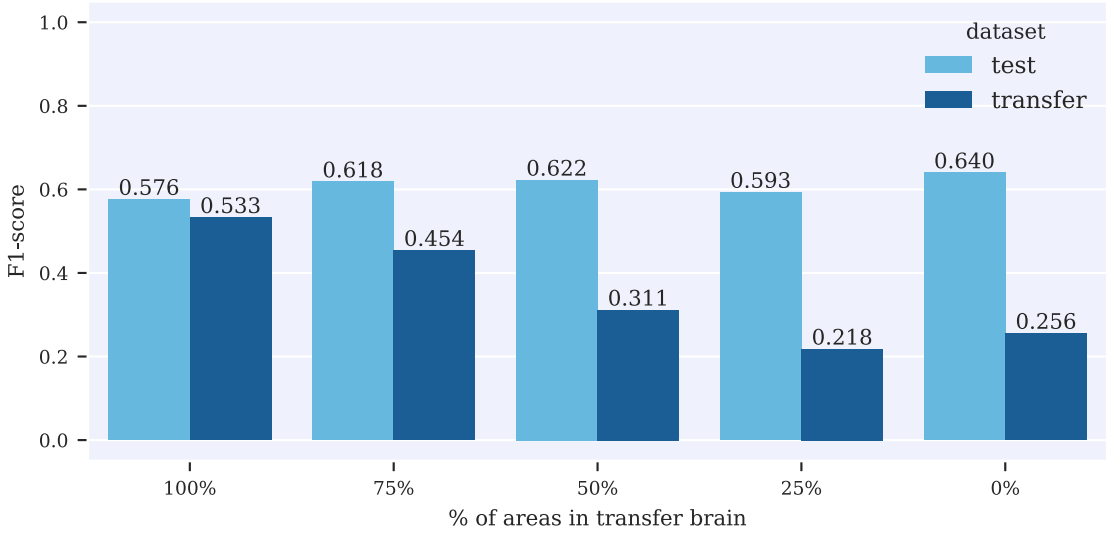



**Figure 6.7:** Test and transfer scores with increasing numbers of training patches sampled from the transfer brain. The percentage indicates the proportion of areas for which image patches from the transfer brain are included during training. In the transfer brain, 240 image patches from each considered brain area are selected for training. Experiments where only a subset of areas from the transfer brain is considered (25%, 50%, 75%) are repeated three times with different random seeds, and average scores across repetitions are reported. Performance on the transfer brain is evaluated on sections from which no training samples are drawn.

This question cannot be answered directly, as "unseen areas" are, by definition, not known when the feature encoder $f_E$ is trained. However, we can simulate the mechanism of acquiring annotations for unseen areas: In the pre-training stage, we randomly remove training samples for increasing proportions of brain areas. The removed areas represent "unknown" areas, which the feature encoder cannot use for learning. The linear classifier in the finetuning stage is trained using *all* training examples (i.e., including samples of previously removed areas), allowing us to estimate how well the learned features predict unknown brain areas.

For our experiments, we randomly remove 25%, 50%, and 75% of the considered brain areas from the training set of the feature encoder. We repeat each experiment with three random seeds to account for the randomness in this step. The performance is compared to a model trained on all available brain areas (100%), representing our default training protocol.

Figure 6.8 shows test and transfer F1-scores when different subsets of brain areas are used for the pre-training. Average scores across runs are reported for repeated experiments.

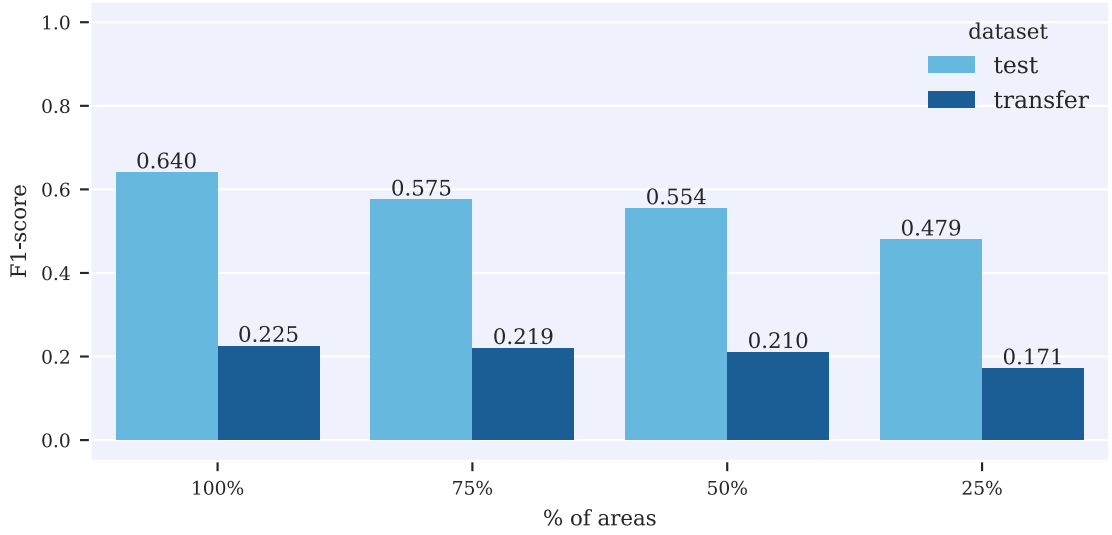



**Figure 6.8:** Test and transfer F1-scores obtained using different subsets of brain areas for contrastive pre-training. Contrastive pre-training is performed with the respective reduced set of brain areas. The following finetuning uses the full set of brain areas.

Test scores decrease as the proportion of brain areas excluded from pre-training increases. There is a noteworthy performance loss when reducing the proportion of brain areas in the training set from 100% to 75% and from 50% to 25%. The performance loss between 75% and 50% is relatively small. Models trained using 75% or 50% of all brain areas outperform the baseline model `R50-CE`, which is trained with categorical cross-entropy and *all* considered brain areas (see Figure 6.4).

Compared to the test scores, there is only a moderate decrease in the transfer scores. The number of considered brain areas shows no considerable effect on the transferability to unseen brains.

The results show that features learned using contrastive learning are useful to classify unseen brain areas. The observed performance loss is moderate regarding the number of areas removed from the training set. Models trained with contrastive learning outperform the baseline model, even when using a significantly reduced number of training examples.

### 6.2.5 Classification performance with varying sample counts

We investigate how the number of training samples per cytoarchitectonic area affects the prediction performance. Insights on the relationship between classification performance and the number of training samples are important for practical applica-

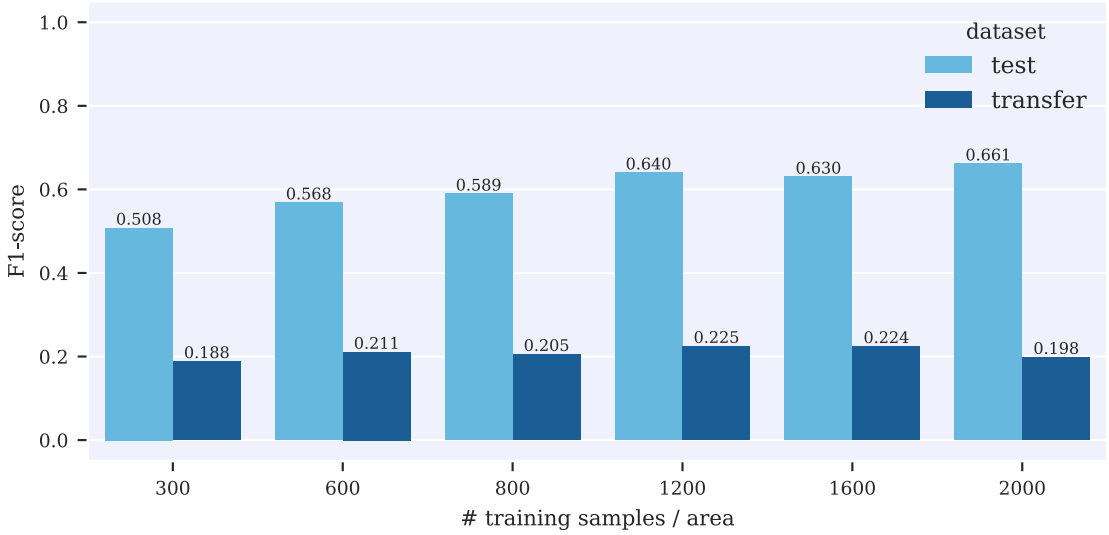



**Figure 6.9:** Test and transfer F1-scores obtained with a varying number of training samples per area. The default dataset is represented by the experiment with 1200 training samples per brain area.

tions. They allow estimating how many annotations should be provided to achieve satisfactory performance.

The default experimental setting includes $n_s = 1200$ image patches from each considered brain area (Section 6.1.2), corresponding to a median sampling ratio $\bar{r}_{\mathcal{A}}$ close to 100% (i.e., areas are neither strongly over- nor undersampled). We vary the sampling rate $n_s$ and investigate the performance with 300, 600, 800, 1600, and 2000 samples per area. These sample counts correspond to median sampling ratios $\bar{r}_{\mathcal{A}}$ of 25%, 50%, 75%, 133%, and 166%, respectively. Figure 6.9 shows the test and transfer F1-scores with varying numbers of training samples per area.

Test scores increase monotonically up to 1200 samples per area. Training with 1600 and 2000 samples per area results in moderately decreased and improved scores compared to the default setting, respectively. Using more than 1200 samples per area increases the median sampling ratio to more than 100%, thus introducing more duplicated training samples. It is plausible that duplicating the training samples does not lead to improved performance[4]. This suggests that determining the sample count per area based on the median sampling ratio is reasonable.

Models trained with 600 or 800 samples per area achieve higher test scores than the baseline model `R50-CE` (Figure 6.4), which is trained with categorical cross-entropy on the *full dataset* (i.e., 1200 images per area).

[4]Including (more) duplicated training samples has a similar effect as training for more epochs.

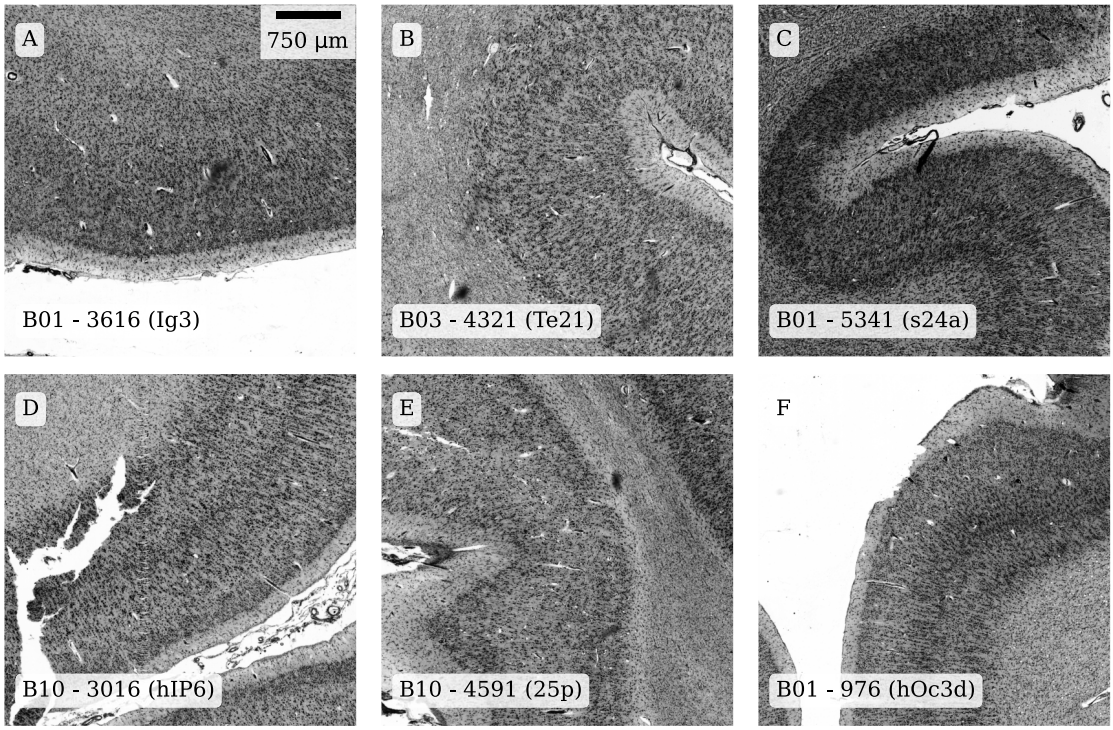



**Figure 6.10:** Randomly selected image patches from test sections $\mathcal{S}_{te}$. Image patches are used to investigate the robustness of trained models to input variations (Figure 6.11). The text in the lower left of each patch indicates the brain, the section number, and the brain area from which the respective patch was extracted. Image D shows an example of tissue that was damaged during histological preparation.

The transfer scores show minimal variations and remain mostly unaffected by the change in training samples. This result indicates that providing more examples from training brains does not improve the classification performance for unseen brains.

### 6.2.6 Robustness to input variations

Understanding how trained models react to variations in the input image patches (e.g., rotation, blurring, intensity variations) helps to assess its robustness to naturally occurring variations. It can also help to investigate potential reasons for the limited applicability of trained models to unseen brains (Section 6.2.3), as transformations to which a model is invariant can be ruled out as potential problems.

We evaluate the robustness to input variations by measuring how the feature vectors produced by the trained encoder $f_E$ of `R50-SCL` change when certain transformations are applied to input images. We randomly select six image patches from the test sections (Figure 6.10). We compute the feature vector $\boldsymbol{h}_i = f_E(X_i)$ for each image patch $X_i$. We then apply transformations with different parameters to each

image patch, use the encoder to compute corresponding feature vectors, and compute the cosine similarity between the feature vectors of unmodified and transformed image patches. Transformations are chosen according to the data augmentation parameters (Section 6.1.3) used during training.

Figure 6.11 visualizes how different image transformations influence learned feature representations. Each subplot shows the similarity between the feature vectors of unmodified image patches (marked by red dots) and transformed images with varying transformation parameters. Similarity scores close to one indicate that the feature vectors of a transformed image and the respective unmodified image are similar and consequently that the model is robust against the specific transformation. As similarity scores alone are difficult to interpret, each subplot shows the maximum and minimum similarity observed between any of the six considered image patches. They illustrate which similarity scores can be expected for two distinct image patches, thereby aiding the interpretation. The similarity matrix for the considered image patches is shown in the bottom right subplot of Figure 6.11.

The results in Figure 6.11 show that the learned feature representations are robust against translations of up to 500 mm in any direction, as well as to arbitrary rotations. The model does not rely on image patches to be placed exactly in the center of the cortex and does not require image patches to be rotated in a specific way. This result is practically relevant, as it can be challenging to ensure consistent placement and rotation of image patches.

Learned representations are also robust against brightness and contrast variations, which is crucial to address staining differences (Figure 3.1). It indicates that the model focuses on patterns in the data (e.g., the distribution of cells) rather than on the specific intensity values.

Strongest variations are observed under blurring and sharpening. In particular, blurring leads to significantly altered feature vectors as the standard deviation (i.e., the size of the Gaussian kernel used for blurring) increases. Strong blurring obstructs the cellular composition and makes recognition of cytoarchitectonic area difficult. Thus, the strong effect of blurring could indicate that the model learns to extract fine-grained cytoarchitectonic patterns.

### 6.2.7 Feature space cluster analysis

Analyzing the feature space of trained deep neural networks can help to understand their behavior and gain an impression of how models "see" the data. In this section, we cluster and visualize feature vectors in the feature space and investigate if clusters correspond to semantically meaningful concepts. Spitzer et al. (2018a) presented a similar feature analysis for the SSL task presented in Spitzer et al. (2018b) (Sec-

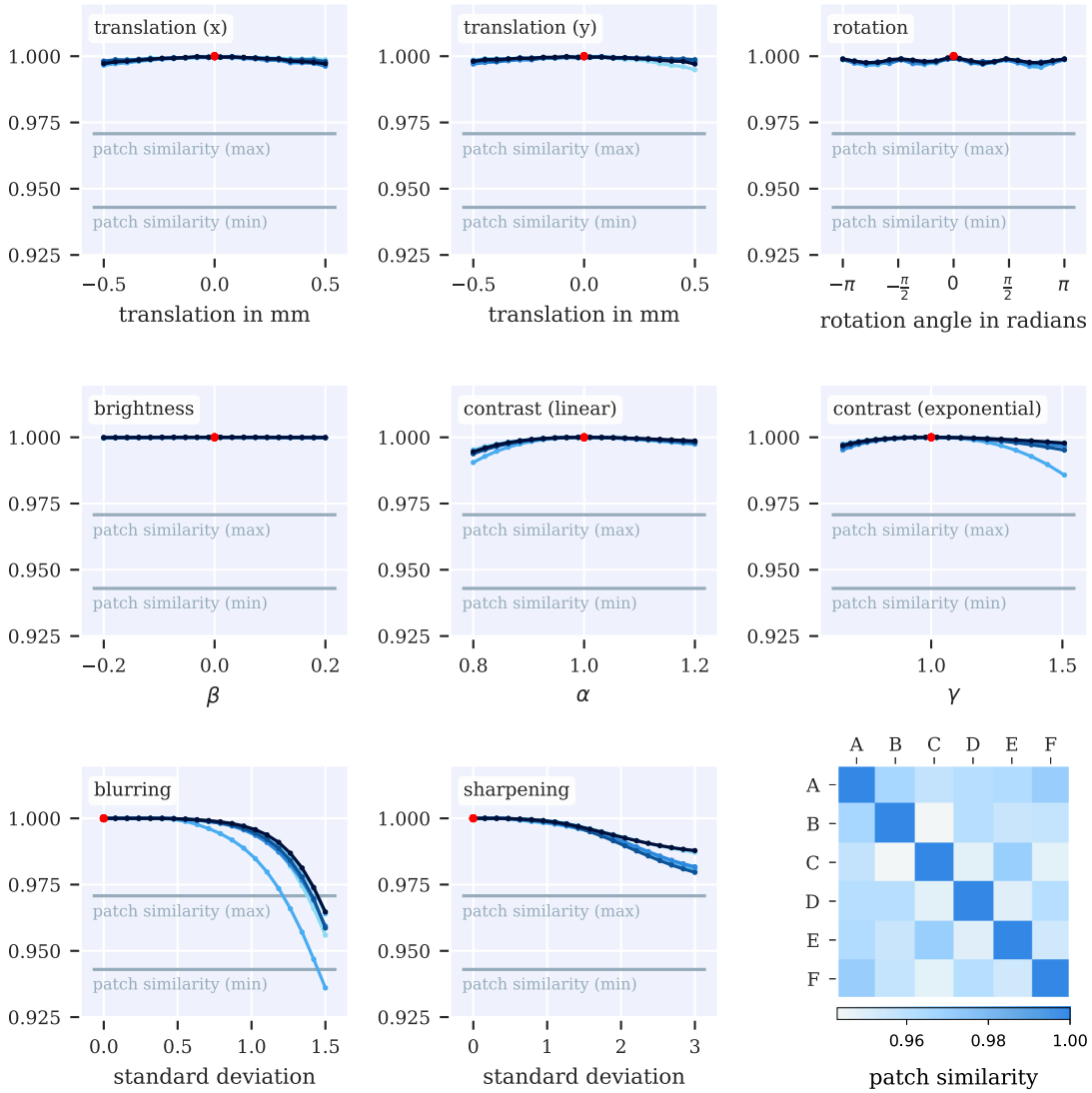



**Figure 6.11:** Similarity of feature vectors under different transformations of the input images (Figure 6.10). The subplots show the similarity between the feature vector of unmodified image patches and feature vectors of transformed images with varying transformation parameters. The similarity between feature vectors is measured using the cosine similarity. Similarities close to one indicate robustness to the respective transformation. Blue lines represent results for different image patches. Red dots mark parameter configurations that do not modify an image patch (i.e., the unmodified images). The similarity matrix visualizes the similarity between the six image patches (Figure 6.10). The minimum and maximum similarity from the similarity matrix are shown in each plot (patch similarity (min/max)). They illustrate the similarity scores than can be expected for two distinct image patches and aid with the interpretation of similarity scores under data transformation.

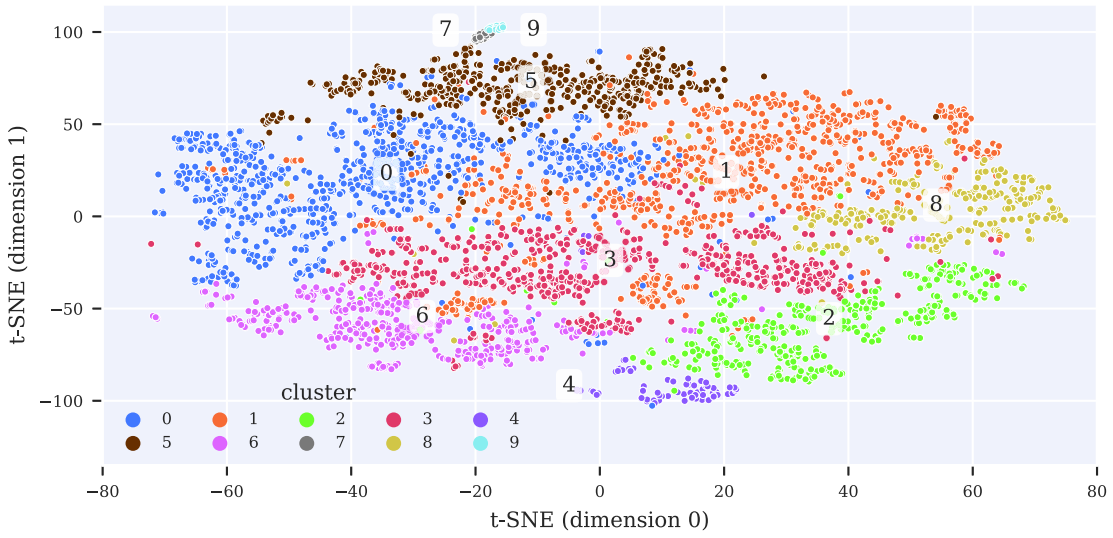



**Figure 6.12:** Two-dimensional t-SNE (Van der Maaten et al., 2008) embedding of feature vectors used by `R50-SCL`. Clusters computed using Ward hierarchical clustering (Ward Jr, 1963) are color-coded.

tion 2.2.7) and showed that the learned representations encode cytoarchitectonic features.

We conduct the analysis based on example image patches from the test sections $\mathcal{S}_{te}$. We extract 50 image patches per cytoarchitectonic area, resulting in a total of 5650 images. The encoder of `R50-SCL` is used to extract feature vectors from the selected image patches. We use PCA to reduce the dimensionality of the feature vectors from 2048 to 256, retaining a total explained variance of 93.6%. The resulting feature vectors are clustered into ten clusters[5] using the Ward hierarchical clustering algorithm (Ward Jr, 1963). We analyze the composition of the clusters, in particular the cytoarchitectonic areas to which samples in each cluster belong. The 60% most frequent cytoarchitectonic areas in each cluster are listed in Table 6.3.

In addition to analyzing the cluster composition, visualizing the feature space can help to better understand its structure. As the high-dimensional feature vectors cannot be visualized directly, we employ the non-linear embedding method *t-SNE* (Van der Maaten et al., 2008) to project the feature vectors into a two-dimensional embedding space. Figure 6.12 visualizes the resulting embeddings with clusters encoded by different colors.

Figure 6.13 additionally visualizes the composition of each cluster with respect to the contained cytoarchitectonic areas. The plot shows the mean locations in the

[5]For visualization purposes, we use a relatively few clusters, but this can be freely adapted for other analyses.

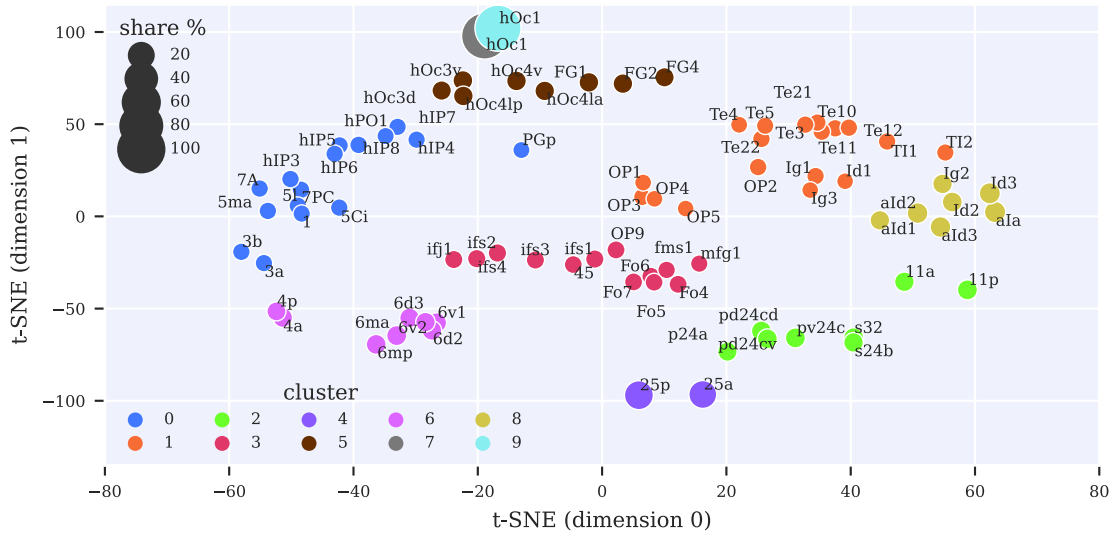



**Figure 6.13:** Composition of feature space clusters with respect to the contained cytoarchitectonic areas visualized in a two-dimensional t-SNE (Van der Maaten et al., 2008) embedding space. Each point corresponds to the mean position of a cytoarchitectonic area within the containing clusters. The point size visualizes the relative share of each area within the respective cluster. Only the 60% most frequent cytoarchitectonic areas per cluster are shown.

t-SNE space for the 60% most frequent areas contained in each cluster (i.e., those shown in Table 6.3). The size of the points in Figure 6.13 visualizes the relative share of each area within the containing cluster.

The composition and position of the identified clusters align well with the underlying anatomical principles and reveal several semantically meaningful patterns. Clusters 7 and 9 both contain exclusively image patches from `hOc1`, the primary visual cortex. Their position is almost identical and separated from the remaining data points. A plausible explanation for this observation is that `hOc1` shows clear and strongly expressed cytoarchitectonic features (Figure 2.5). This hypothesis is supported by the fact that clusters 7 and 9 are much denser than other clusters, indicating that the features defining `hOc1` show little variance.

Cluster 5, which lies closest to clusters 7 and 9 in the t-SNE space, primarily comprises areas from the visual system. Thus, the composition and location of the cluster correspond well with the underlying anatomical concept. Similar effects can be observed in several of the other clusters: Cluster 0 contains areas from the parietal lobe, which is adjacent to the occipital lobe that contains the visual areas. Within cluster 0, we observe a gradient from areas located anatomically close to the occipital lobe (e.g., `hPO1`, `hIP7`, `PGp`) to areas that are further away (e.g., `5ma`, `7PC`). Somatosensory areas `3a` and `3b` at the border of cluster 0 are close to primary motor areas `4a` and `4p`

**Table 6.3:** Frequency of cytoarchitectonic areas observed in different feature space clusters. Only the 60% most frequent cytoarchitectonic areas per cluster are shown.

| | | | | |
|---|---|---|---|---|
| 0 | hIP8 (4.1%) | pc1 (4.1%) | mip (4.1%) | hIP5 (4.0%) |
| | 5l (4.0%) | 5ma (3.9%) | 5mb (3.9%) | pc2 (3.9%) |
| | hIP6 (3.8%) | 3b (3.8%) | hIP7 (3.8%) | hIP4 (3.8%) |
| | 3a (3.8%) | eins (3.6%) | hPO1 (3.3%) | PGp (3.1%) |
| 1 | Te10 (4.0%) | Te11 (4.0%) | OP2 (4.0%) | Te22 (4.0%) |
| | Te12 (4.0%) | TI1 (3.9%) | Te21 (3.9%) | Ig1 (3.8%) |
| | Te3 (3.8%) | TI2 (3.6%) | Te5 (3.6%) | OP3 (3.4%) |
| | Te4 (2.9%) | OP1 (2.8%) | OP4 (2.6%) | Ig3 (2.5%) |
| | Op5 (2.5%) | Id1 (2.5%) | | |
| 2 | s32 (8.8%) | 11p (8.6%) | pv24c (8.6%) | pd24cd (8.4%) |
| | pd24cv (8.2%) | s24b (7.7%) | 11a (7.7%) | p24a (7.2%) |
| 3 | ifs4 (5.5%) | ifs2 (5.5%) | ifs1 (5.4%) | ifj1 (5.2%) |
| | ifs3 (5.2%) | Fo4 (4.8%) | Op9 (4.8%) | Fo7 (4.7%) |
| | Fo6 (4.6%) | 45 (4.5%) | Fo5 (4.5%) | fms1 (3.9%) |
| | mfg1 (3.6%) | | | |
| 4 | 25p (31.7%) | 25a (28.5%) | | |
| 5 | hOc3v (8.1%) | FG1 (8.0%) | hOc4v (8.0%) | hOc4lp (8.0%) |
| | hOc4la (7.8%) | FG2 (7.8%) | hOc3d (7.5%) | fg4 (7.3%) |
| 6 | presma (8.7%) | sma (8.3%) | 6v1 (8.1%) | 4a (7.8%) |
| | 6d2 (7.8%) | 6d3 (7.8%) | 6v2 (7.6%) | 4p (6.7%) |
| 7 | hOc1 (100.0%) | | | |
| 8 | aIa (11.3%) | Id3 (11.1%) | aId3 (10.1%) | aId2 (10.1%) |
| | Id2 (8.5%) | aId1 (7.7%) | Ig2 (7.7%) | |
| 9 | hOc1 (100.0%) | | | |

in cluster 6, which are located spatially close in the brain. Within cluster 6, premotor (e.g., 6d2, 6d3, 6v1, 6v2) and supplementary motor areas (e.g., 6ma, 6mp) form a sub-cluster. Cluster 1 contains a range of areas from the temporal lobe, cluster 3 contains frontal areas, cluster 8 contains areas from the insula, and clusters 2 and 4 contain areas from the limbic lobe. In many cases, the relative position of clusters to each other and the average position of areas within the clusters correspond well with structural brain organization.

### 6.2.8 Limitations of self-supervised contrastive learning for cytoarchitectonic mapping

Self-supervised contrastive learning methods (Chen et al., 2020) can learn visual representations from large image datasets without relying on supervision through class labels (Section 2.2.5). In comparison, the supervised contrastive learning approach by

Khosla et al. (2020) and our adapted supervised contrastive learning approach (Section 6.1.1) rely on the availability of class labels to define the similarity between image samples. Here, we provide the rationale behind the decision to use a supervised approach, despite its reliance on labeled training examples.

We investigate the self-supervised SimCLR (Section 2.2.5) for cytoarchitecture classification. Training is performed as described in Section 6.1.4. The only difference is the loss function, as we use the self-supervised contrastive loss (Equation 2.53, Chen et al., 2020) rather than the supervised contrastive loss (Section 6.1.1). The self-supervised contrastive loss considers views of the same image as similar. We use the data augmentation operations described in Section 6.1.3 to create two views from each image patch in a batch.

Self-supervised training converges quickly to low loss values, indicating that the trained model finds a good solution to the posed pre-training task. However, in the finetuning stage, the model achieves poor linear evaluation performance, often not better than a random classifier that assigns labels by chance. Although the model learns to solve the contrastive pre-training task (i.e., matching views of images), the features learned in the process are not useful for the downstream task (i.e., classifying cytoarchitectonic areas). We observe a similar behavior using the supervised contrastive loss (Equation 2.59) by Khosla et al. (2020), which regards two views as similar if they originate from the same source image *or* if they share the same class label.

To understand the cause of the observed behavior, we investigate which images are considered "similar" by the trained SimCLR model. The first column in Figure 6.14 shows three example image patches from the test sections $\mathcal{S}_{te}$ (A, B, C). For each of these three reference patches, we identify the four image patches that the model considers most similar to the respective reference patch, measured by the cosine similarity between feature vectors.

The comparison suggests that the model defines similarity based on anatomical landmarks and other macroscopic features:

- **Row A**: Several small microstructures (e.g., blood vessels), which appear as white dots in the image.
- **Row B**: Tissue with a characteristic folding pattern.
- **Row C**: Relatively straight portion of the cortex without strong folding.

A comprehensive study of more example patches shows similar reoccurring patterns in many image patches.

The results suggest that the SimCLR models exploit macroscopic rather than microstructural (i.e., cytoarchitectonic) features. These macroscopic features are suf-

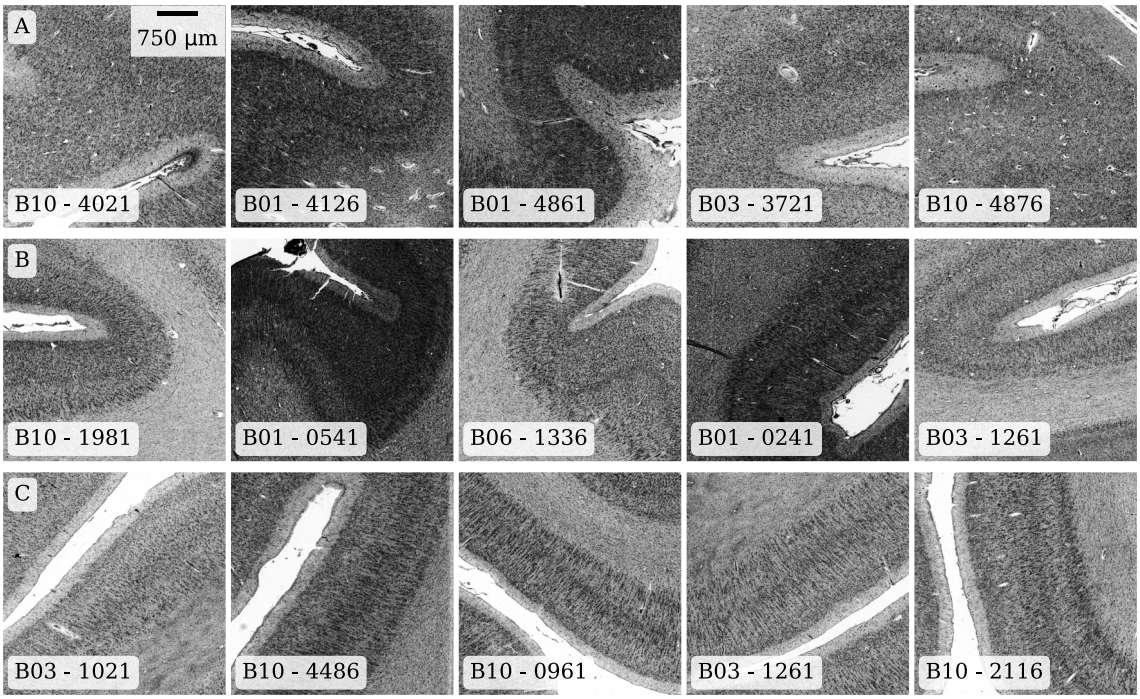



**Figure 6.14:** Image patches from the cortex that are considered similar by a model trained using SimCLR (Chen et al., 2020). Images in the left column represent reference patches. Each row shows the patches that are most similar to the respective patch in the first column. The similarity is measured by the cosine similarity between the feature vectors of each image patch.

ficient to solve the contrastive pre-training task, but not useful for cytoarchitecture classification.

An extended discussion of the role of self-supervised contrastive learning methods for cytoarchitecture classification is given in Section 8.2.

## 6.3 Summary

In this chapter, we introduced a supervised contrastive learning method (Section 6.1.1) for automated classification of many cytoarchitectonic areas across different brains. The presented method builds upon earlier work on automated classification of visual cytoarchitectonic areas (Spitzer et al., 2017, 2018b) and represents an important improvement towards automated cytoarchitectonic mapping at a whole-brain scale.

Supervised contrastive learning outperforms baseline models trained with categorical cross-entropy (Sections 6.2.1 and 6.2.2), while being more efficient with respect to training data requirements (Sections 6.2.4 and 6.2.5). Features extracted by trained

models are robust against natural variations in the data (Section 6.2.6) and encode anatomically meaningful properties (Section 6.2.7).

We identified the limited applicability of trained models to unseen brains (Section 6.2.4) as a major remaining challenge. Finally, our results on self-supervised contrastive learning (Section 6.2.8) show that approaches that yield good results for other image classification tasks (Chen et al., 2020) are not necessarily applicable for cytoarchitecture classification.

A comprehensive discussion of the presented results is given in Section 8.2.

# 7 Topology-aware cytoarchitectonic mapping with graph neural networks

This chapter introduces a method for automated cytoarchitecture classification using GNNs, which enables the incorporation of contextual information into the classification process. Existing approaches model brain mapping as segmentation (Spitzer et al., 2017, 2018b, Chapter 5) or classification problem (Chapter 6) on individual high-resolution image patches. High-resolution image patches allow extraction of local cytoarchitectonic features, but they do not capture information on 3D brain topology (i.e., the neighborhood relationship between image patches) or the 3D location of image patches in the brain. Integrating such information is often crucial to disambiguate and correctly classify cytoarchitectonic areas.

Here, we combine cytoarchitectonic features with 3D brain topology by formulating cytoarchitecture classification as a node classification problem in a graph representing the cortical brain surface (Section 7.1.1). We apply the contrastive learning method presented in Chapter 6 to encode cytoarchitectonic features from high-resolution 2D image patches into compact feature vectors. Created feature vectors are assigned to nodes of a graph representing the 3D surface of the brain. This allows using GNNs to integrate local cytoarchitectonic features from high-resolution image patches with information on 3D brain topology encoded in the graph.

The workflow for constructing attributed graphs of the cortical brain surface is described in Section 7.1.1. We investigate the performance of the proposed method with different GNN architectures (Section 7.1.4) and compare it to the contrastive learning method presented in Chapter 6 (Section 7.2.1). In addition, we examine how the incorporation of additional neuroanatomical priors into the classification framework affects the performance of the method (Section 7.2.2).

The method reported in this chapter was published in the article "2D Histology Meets 3D Topology: Cytoarchitectonic Brain Mapping with Graph Neural Networks" (Schiffer et al., 2021c).

# 7.1 Methods

## 7.1.1 Brain mapping as node classification problem

We model brain mapping as a node classification task in graphs that approximately model the 3D topology of the brain surface. We compute such graphs by creating approximate *midsurface meshes* through the cortex. A midsurface mesh is a representation of the cortical surface passing through the midpoints between pial boundary and gray-white matter boundary, which is computed from an approximate 3D reconstruction of a brain.

The created mesh is interpreted as a graph consisting of nodes and connecting edges, which we refer to as *cortical midsurface graph*. Using information from the 3D reconstruction step, each node in the graph can be associated with a location in a histological brain section, enabling the assignment of information from the 2D image domain (e.g., cytoarchitectonic features) to graph nodes. The attributed midsurface graphs encode 3D brain topology and 2D image features in a computationally efficient data structure.

The following sections describe the workflow for constructing cortical midsurface graphs from histological brain sections. It comprises the following steps:

- Approximate **3D reconstruction** from histological brain sections (Section 7.1.1).
- Cortical **midsurface graph extraction** (Section 7.1.1).
- **Identification** of 3D graph nodes with 2D image locations (Section 7.1.1).
- **Feature assignment** to graph nodes (Section 7.1.1).

### Approximate 3D brain reconstruction

We approximately recover the 3D structure of the brains to create midsurface graphs through the cortex. We use a rigid alignment of consecutive brain sections (Section 3.2) to assemble consistent 3D volumes from individual brain sections. The rigid transformations enable us to approximately align each section to its respective adjacent sections. For each brain, we define a *base section* in the center of the section stack. All other sections of a brain are then aligned to the base section through recursive application of the rigid transformations (Section 3.2). Limiting the alignment to rigid transformations (i.e., translation and rotation, but no scaling) avoids strong deformations and distorted morphology of the resulting reconstructed brain volumes (e.g., the "banana effect", Malandain et al., 2004). Figure 7.1 shows the result of the 3D reconstruction by the example of `B03`.

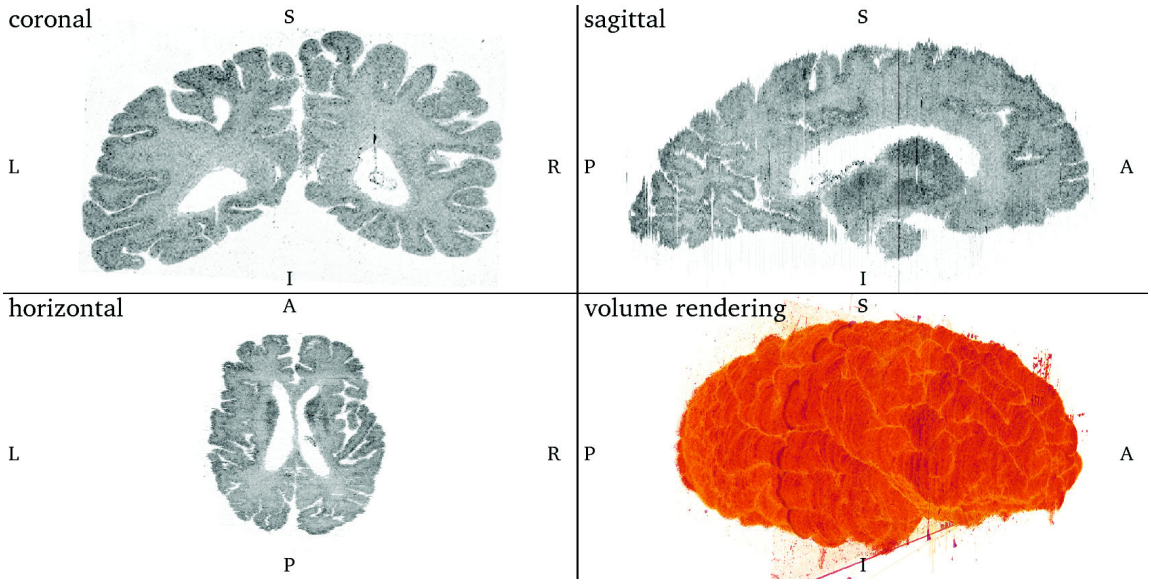



**Figure 7.1:** Approximate 3D reconstruction of B03 created from aligned histological brain sections. The coronal plane represents the original sectioning plane. The horizontal and sagittal planes are virtual cross-sections through the reconstructed volume. The cross-sections show that the three-dimensional structure is approximately recovered by the reconstruction. The lower right pane shows a volume rendering of the reconstructed volume. Letters indicate anatomical directions (L/R: left/right, P/A: posterior/anterior, I/S: inferior/superior). Resolution: 300 µm/vx (isotropic).

### Cortical midsurface graph computation

Computing a cortical midsurface graph requires a cortex segmentation, i.e., a mask indicating which voxels belong to the cortex. We create a cortex segmentation by applying the described reconstruction pipeline to the tissue segmentation described in Section 3.2. This reconstruction step results in one 3D segmentation volume per brain, where each voxel is labeled as background (i.e., non-tissue), white matter, or gray matter (Figure 7.2, A).

Reconstruction and post-processing of the segmentation volumes are performed at an isotropic resolution of 300 µm/vx. Segmentation masks are downscaled using nearest neighbor interpolation before reconstruction. The resolution is determined by the availability of digitized histological sections, as typically every 15th brain section is digitized (Section 3.2). Each sections has a thickness of 20 µm, resulting in a resolution of 300 µm in the posterior-anterior direction. The resolution is sufficient to accurately identify the cortex in the reconstructed segmentation volumes.

We clean created segmentation volumes to remove errors resulting from histological artifacts, incorrect tissue segmentations, or imprecise alignment of consecutive sections. Tissue defects are detected by smoothing the volume with a median fil-

ter (size 3) along the posterior-anterior direction and computing the difference to the input volume. Large connected components in the difference volume represent large tissue defects, which are replaced by the result of the median filter. Correcting large defects ensures a consistent morphology of the reconstructed cortex by "filling" gaps. We then remove small parts of detached tissue by removing connected tissue components smaller than 1% of the largest tissue component (Figure 7.1, B).

The tissue segmentation typically assigns subcortical gray matter and the cerebellum to the cortex (Figure 7.2, B), which is expected because the tissue segmentation is based on pixel intensities (Section 3.2). Since our analysis of cytoarchitectonic areas is restricted to the cortex, excluding such incorrectly classified voxels improves the quality of the created meshes. We use the software *3DSlicer* (Kikinis et al., 2014) to create masks of subcortical gray matter and the cerebellum. The software allows drawing masks on a few cross-sections of a volume, which are then interpolated to a full volumetric mask. This step requires approximately 30 min per brain. The created masks are used to remove all gray matter voxels that do not belong to the cortex (Figure 7.2, C).

In the next step, the cortical midsurface is computed from each prepared 3D segmentation volume. The cortical midsurface is defined as the surface in the center between pial boundary and gray-white matter boundary. Following the approach presented in Leprince et al. (2015), we compute a *Laplacian field* in the cortex using the software *BrainVisa* (Rivière et al., 2009). The Laplacian field approximates the cortical depth (Figure 7.2, D). It takes a value of 0.0 at the pial boundary and linearly increases to 1.0 towards the gray-white matter boundary. We then apply the *marching cubes algorithm* (Lewiner et al., 2003) to extract the 0.5-isosurface from the Laplacian field. The extracted isosurface represents the midsurface through the cortex.

We clean each midsurface mesh by removing small isolated connected components, splitting brain hemispheres into separate meshes, fixing topological errors in the mesh, and computing a *Poisson surface reconstruction* (Kazhdan et al., 2006) to remove artifacts resulting from inaccuracies in the tissue segmentation or the reconstruction step (Figure 7.1). We then apply *isotropic explicit remeshing* (Surazhsky et al., 2003) to ensure that all triangles in a mesh have an edge length of approximately 300 µm. Remeshing reduces the number of triangles and ensures that connections between vertices in the mesh represent comparable distances. We use 300 µm as target edge distance because it corresponds to the expected distance between two adjacent sections (thickness: 20 µm) in a brain where every 15th section is scanned (Section 3.2). Mesh processing is performed using the software *MeshLab* (Cignoni et al., 2008).

We visually inspect the created meshes to assess their quality (Figure 7.3). Meshes approximately recover the morphology of the brains and encode the neighborhood re-

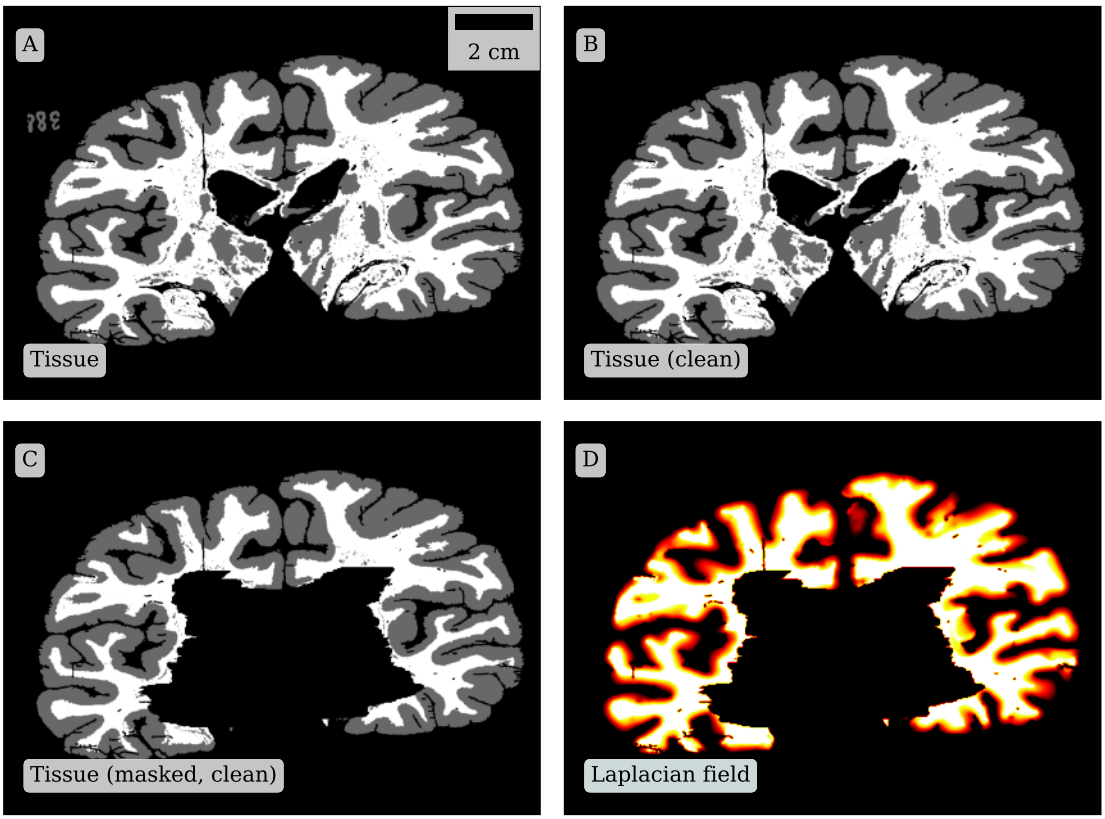



**Figure 7.2:** Cortical midsurface computation for a coronal cross-section through the reconstructed segmentation volume of B03. All steps are performed in 3D, but only a single cross-section is shown here for illustration purposes. **A**: Tissue segmentation before cleaning (black: background, white: white matter, gray: gray matter). **B**: Tissue segmentation after cleaning. **C**: Tissue segmentation after removal of subcortical gray matter and the cerebellum. **D**: Laplacian field computed in the cortex, which can be interpreted as the approximate cortical depth.

lationship between locations in the brain. The linear reconstruction workflow cannot account for strongly deformed or damaged tissue, resulting in some smaller artifacts like holes or "noses" that are visible in the meshes (Figure 7.3). Observed artifacts represent defects or deformations that are present in the histological sections and thus do not negatively affect the GNN method.

The meshes do not need to be anatomically highly plausible, as the relative relationship between mesh vertices is more important than their precise absolute positioning. They are thus not comparable to high-resolution 3D brain reconstructions like the BigBrain model (Amunts et al., 2013), which aim to precisely recover a brain's original morphology. Using only approximate brain reconstructions enables us to use simpler reconstruction workflows that are easily applicable in practice.

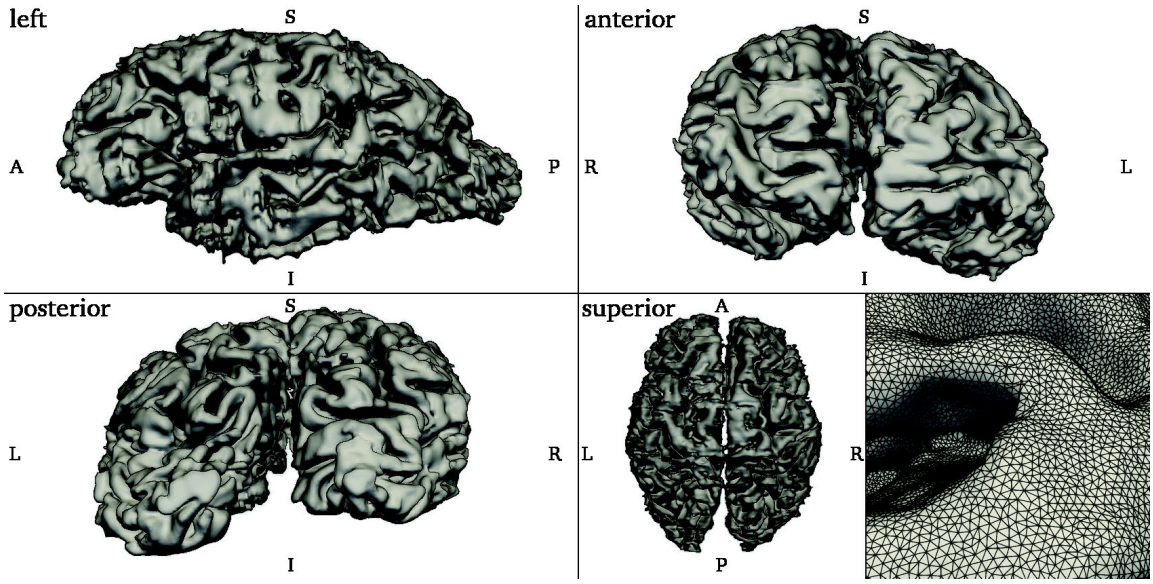



**Figure 7.3:** Midsurface through the cortex of B03 from different perspectives. The figure shows meshes of both brain hemispheres after cleaning, which involves removing isolated connected components, fixing topological errors, and Poisson surface reconstruction (Kazhdan et al., 2006). The closeup in the bottom right pane shows that the mesh is composed of triangles with an approximately identical edge length of 300 µm, which is ensured by isotropic explicit remeshing (Surazhsky et al., 2003). Letters indicate anatomical directions (L/R: left/right, P/A: posterior/anterior, I/S: inferior/superior). Visualization created using *ParaView* (Ahrens et al., 2005).

We interpret the created meshes as graphs. Mesh vertices are interpreted as graph nodes, and the edges of triangles are interpreted as edges in the graphs. We denote the graph representing the cortical midsurface of a brain B as $\mathcal{G}_{\mathtt{B}} = (\mathcal{V}_{\mathtt{B}}, \mathcal{E}_{\mathtt{B}})$ with nodes $\mathcal{V}_{\mathtt{B}}$ and edges $\mathcal{E}_{\mathtt{B}}$[1].

### Identifying 3D graph nodes with 2D image locations

We aim to assign features defined in the image domain (i.e., brain section images) to corresponding nodes in the created midsurface graphs. This step allows combining 2D image features with 3D brain topology. We identify the corresponding location in one of the brain section images for each node in a midsurface graph.

For each brain section, we determine a 2D plane that cuts through the midsurface graph at the location of the respective brain section (Figure 7.4, left). The location of the plane is obtained from the 3D reconstruction pipeline (Section 7.1.1). We compute the intersection between the plane and the midsurface graph. This operation

[1] Technically, each brain results in one graph *per hemisphere*. However, for the sake of brevity, we omit the dependence on the hemisphere from the notation and model each brain as a single graph with multiple connected components.

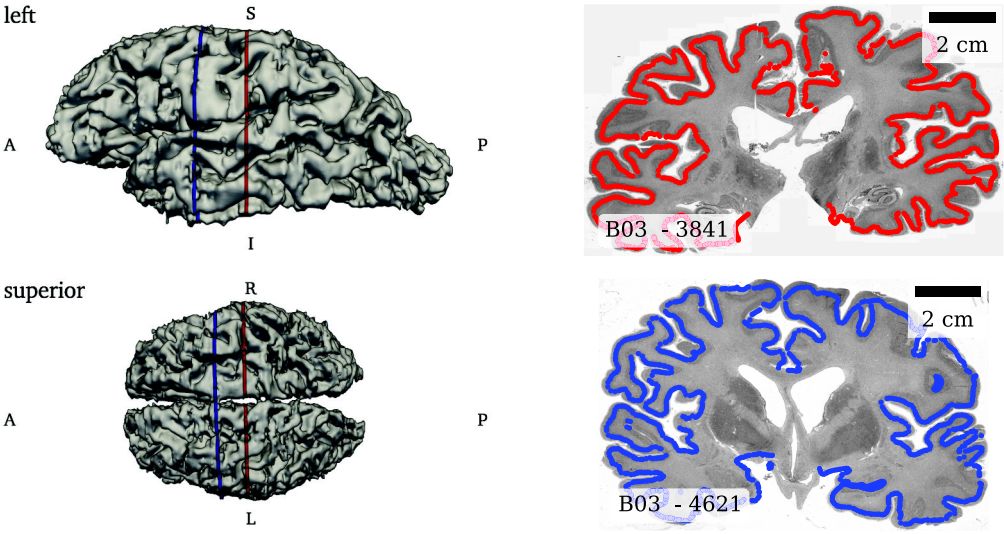



**Figure 7.4:** Correspondence between nodes in the midsurface graph of B03 (left) and points in two histological brain sections (right). Colored lines indicate the location of the two shown brain sections within the midsurface graph of the cortex. The midsurface graph is virtually cut at the indicated positions to obtain corresponding points in the sections shown on the right. Letters indicate anatomical directions (L/R: left/right, P/A: posterior/anterior, I/S: inferior/superior). Visualization created using *ParaView* (Ahrens et al., 2005).

can be interpreted as "virtually cutting" the reconstructed brain. The point set resulting from the intersection is transformed back onto the given brain section by inverting the 3D reconstruction workflow. The transformation step leaves us with points defined on the brain section image. Each of the points can then be uniquely identified with a node in the midsurface graph (Figure 7.4, right). This relationship is used to compute features in the image domain and assign them to the corresponding location in the midsurface graph.

The transformed points are not always located along the midline through the cortex. These distortions result from smoothing and cleaning operations applied during the reconstruction pipeline, which ensure the consistency of the created meshes. We correct the positions of the points in a *refinement step*, which "pushes" points towards the midline of the cortex. For each brain section, we compute the midline of the cortex from the morphological skeleton of the cortex segmentation (Section 3.2). We then compute a Laplacian field (Leprince et al., 2015) between the midline and the pial boundary or gray-white matter boundary using successive over-relaxation. We use a numerical solver for ordinary differential equations to integrate each point through the gradient field of the Laplacian field. This step gradually moves each point towards the midline of the cortex. Compared to simply moving each point

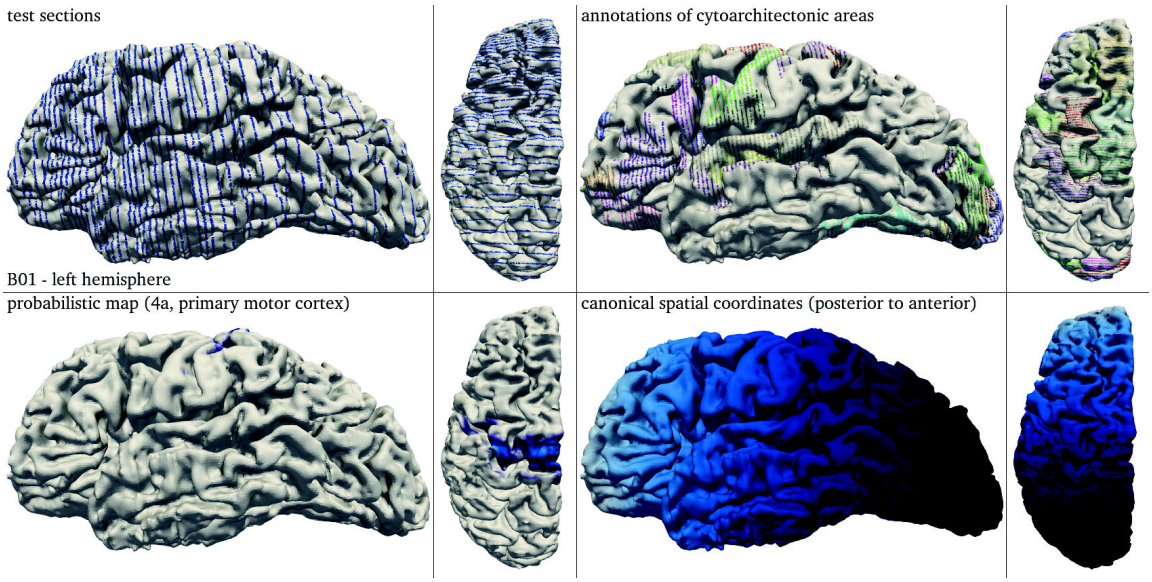



**Figure 7.5:** Example node features assigned to the midsurface graph of the left hemisphere of B01. **Top left**: Distribution of test sections on the midsurface graph. Colors indicate the locations of sections that are used for testing. **Top right**: Annotations of cytoarchitectonic areas. Annotations are encoded by different colors. The availability of annotations in a sparse set of sections results in a visible stripe pattern. **Bottom left**: Probabilistic maps of the primary motor area 4a located on the precentral gyrus. Blue color indicates a high occurrence probability. **Bottom right**: Canonical spatial coordinates defined in the MNI-Colin27 space. The example shows the posterior-anterior component, which increases from low values (dark) in the posterior part of the brain to high values (bright) in the anterior part. Visualization created using *ParaView* (Ahrens et al., 2005).

to the respective closest point on the midline, this more elaborate approach better preserves the spatial distribution of the points along the cortex and leads to better results. We ensure that the correspondence between points in the brain sections and the respective nodes in the midsurface graph is preserved by limiting the maximal movement of each point to 2 mm (approximately half of the cortical thickness).

Despite this refinement step, some points are still located outside the tissue, for example, when the tissue was damaged during histological processing. We identify such points by extracting an image patch centered at each point from the tissue segmentation mask (Section 3.2) and determine the share of tissue within the patch. Points with a tissue share below 50% are considered outside the tissue. These points are excluded from further processing (e.g., training).

### Assigning image features to graph nodes

The known correspondence between locations in the midsurface graph and the image domain allows assigning features from the image domain to nodes of the midsurface graphs and vice versa (Figure 7.5). We consider the following types of image features:

**Deep cytoarchitectonic features** *Deep cytoarchitectonic features* are computed from high-resolution image patches using a deep neural network. We use the `R50-SCL` model presented in Section 6.2.2, which maps square image patches with size 2048 px at 2 µm/px to 2048-dimensional feature vectors. We showed in Chapter 6 that the model learns to extract semantically meaningful cytoarchitectonic features, making it well suited to provide node features for midsurface graphs. Deep cytoarchitectonic node features for brain `B` are denoted as $\mathcal{F}^{\text{cy}}_{\mathcal{V}_\text{B}} = \left\{ \boldsymbol{x}^{\text{cy}}_{v_i} \in \mathbb{R}^{2048} \mid v_i \in \mathcal{V}_\text{B} \right\}$.

**Canonical coordinates** The topology of the midsurface graph provides information about the relationship between locations in the brain. However, the topology only provides information on the *relative* relationship between nodes: A connection between two nodes indicates that the nodes are spatially close to each other[2], but it does not provide information about the *absolute* location of the nodes within the brain. Access to the absolute position (e.g., whether a node is located in the front or back of the brain) within the brain provides valuable additional information. In existing GLI-based brain mapping workflows (Section 2.1.4), such information is used by estimating the approximate location of a brain region based on the section number or known anatomical landmarks.

We explicitly add coordinate vectors as node features to the midsurface, which provide the approximate location of each node with respect to a common canonical reference space. Coordinate vectors must refer to a common reference space to make them comparable across brains and ensure that two nodes from different brains with identical coordinate vectors can be identified as approximately corresponding locations in the reference space. We use the MNI-Colin27 space as a reference space. As described in Section 3.2, we first project a coordinate grid from MNI-Colin27 space onto the histological sections. We then sample the projected coordinate grid at the locations corresponding to nodes of the midsurface graphs and assign the 3D coordinate vectors as node features. Canonical coordinates for nodes of brain `B` are denoted as $\mathcal{F}^{\text{co}}_{\mathcal{V}_\text{B}} = \left\{ \boldsymbol{x}^{\text{co}}_{v_i} \in [-1, +1]^3 \mid v_i \in \mathcal{V}_\text{B} \right\}$.

**Probabilistic maps** Probabilistic maps (Section 2.1.3) encode the probability of a particular brain area being present at a specific location in the brain (Amunts et al.,

[2] The isotropic explicit remeshing step described in Section 7.1.1 ensures that two connected nodes have a distance of approximately 300 µm.

2020). Spitzer et al. (2017) showed that incorporating probabilistic maps in form of a *probabilistic atlas prior* can improve classification performance (Section 2.2.7). We adopt this idea and provide probabilistic maps as additional node features in the midsurface graphs.

The probabilistic atlas prior of Spitzer et al. (2017) only includes probabilistic maps of areas that are considered for classification. However, other probabilistic maps can also provide valuable information, for example, by indicating that *another* area is likely to occur at a specific location. Thus, we use *all* available probabilistic maps to create node features.

We use the workflow described in Section 3.2 to project 152 probabilistic maps from Julich-Brain onto the histological brain sections. We then assign a 152-dimensional vector representing the discrete probability distribution over 152 brain areas to each node in the midsurface graph. Probabilistic node features for brain B are denoted as $\mathcal{F}^{\text{pm}}_{\mathcal{V}_{\text{B}}} = \left\{ \boldsymbol{x}^{\text{pm}}_{v_i} \in [0,1]^{152} \mid v_i \in \mathcal{V}_{\text{B}} \right\}$.

**Annotations of cytoarchitectonic areas** Annotations of cytoarchitectonic areas (Section 3.3) can be interpreted as another kind of node feature. They are defined in the image domain (e.g., in the form of contours or pixel-level segmentations), so they can be assigned to the corresponding nodes in a midsurface graph. Compared to the other mentioned features, which can be sampled at arbitrary locations in the cortex, annotations of brain areas can only be sampled where they are annotated. Thus, only a sparse set of nodes in a midsurface graph gets assigned a vector encoding to which cytoarchitectonic area a respective node belongs, while the brain area of all other nodes is considered unknown. Annotated nodes are used for supervised training of GNNs. One-hot encoded annotations for cytoarchitectonic areas of nodes for brain B are denoted as $\mathcal{F}^{\boldsymbol{y}}_{\mathcal{V}_{\text{B}}} = \left\{ \boldsymbol{y}_{v_i} \in \{0,1\}^{113} \mid v_i \in \mathcal{V}_{\text{B}} \right\}$.

## 7.1.2 Dataset preparation

The dataset preparation follows the protocol used for the contrastive learning described in Section 6.1.2. We consider 113 cytoarchitectonic areas (Table 3.2) and eight brains $\mathcal{B} = \{\text{B01}, \text{B03}, \text{B04}, \text{B05}, \text{B06}, \text{B07}, \text{B10}, \text{B12}\}$ (Table 3.1). Brain B20 is not included, but the method is also compatible with B20 or comparable datasets. Midsurface graphs for each brain are created using the workflow described in Section 7.1.1, resulting in a total of 16 graphs (one graph per hemisphere, i.e., two graphs per brain). On average, each graph consists of 760 000 nodes and 4.5 million edges, with an average node degree of 6. We assign deep cytoarchitectonic features ($\mathcal{F}^{\text{cy}}_{\mathcal{V}_{\text{B}}}$), canonical coordinates ($\mathcal{F}^{\text{co}}_{\mathcal{V}_{\text{B}}}$), probabilistic maps ($\mathcal{F}^{\text{pm}}_{\mathcal{V}_{\text{B}}}$), and annotations of cytoarchitectonic areas ($\mathcal{F}^{\boldsymbol{y}}_{\mathcal{V}_{\text{B}}}$) to the nodes of each graph (Section 7.1.1).

Following the approach described in Section 6.1.2, we define brains for training and testing $\mathcal{B}_{tt} = \mathcal{B} \setminus \{B_h\}$ with sections $\mathcal{S}_{tt}$, and a transfer brain $B_h = \texttt{B07}$ with sections $\mathcal{S}_h$. We split sections $\mathcal{S}_{tt}$ into sections for training $\mathcal{S}_{tr}$ and testing $\mathcal{S}_{te}$. Only graph nodes corresponding to training sections are used for training. Nodes that correspond to test sections are used to evaluate classification performance for unknown sections of known brains (Section 7.1.1). Nodes from the transfer brain are used to evaluate classification performance in unknown brains. We use the same split into training, test, and transfer sections as in Chapter 6, and F1-scores for performance evaluation (i.e., test and transfer F1-scores) are computed at the same locations as in Chapter 6. This makes the results from this chapter comparable to those from Chapter 6 and ensures that none of the data that was used to train the feature extraction model `R50-SCL` is here used for testing.

### 7.1.3 Training parameters

We train each GNN for 100 epochs using categorical cross-entropy as loss function. One epoch corresponds to one pass through all labeled training nodes. Training is performed using the LARS optimizer (Section 2.2.1), Nesterov momentum (Section 2.2.1, Sutskever et al., 2013) with $\mu = 0.9$, and weight decay of $\omega = 0.0001$ for all non-bias parameters. The trust parameter of LARS is set to $\eta = 0.02$ (Equation 2.11).

We use a batch size of $b = 2048$ nodes per batch. For GNNs, the batch size refers to the number of nodes for which an output (e.g., a classification) is computed (Equation 2.44). However, the number of nodes that is processed per batch can be considerably larger, as a GNN also considers the neighborhood of each node in the batch (Figure 7.6). The learning rate is set to $0.001\frac{b}{256} = 0.008$ and is kept constant during training.

Training is implemented using ATLaS (Section 4.2). GNN training is implemented using `PyTorch-Geometric` (Fey et al., 2019). The training of one GNN uses two compute nodes of the JURECA-DC HPC system with a total of eight GPUs. Training is distributed across GPUs using data-parallel DDL. Depending on the depth and complexity of the used GNN layers, training takes between 20 min and 2 h. We use AMP to reduce the memory footprint during training.

### 7.1.4 Network architectures

We train GNNs to classify the cytoarchitectonic area of each node in a cortical midsurface graph, resulting in a node classification task (Section 2.2.2). Nodes with available annotations of cytoarchitectonic areas (Section 7.1.1, $\mathcal{F}_{\mathcal{V}}^{\boldsymbol{y}}$) are used to supervise the training with categorical cross-entropy loss. We investigate GNN architectures com-

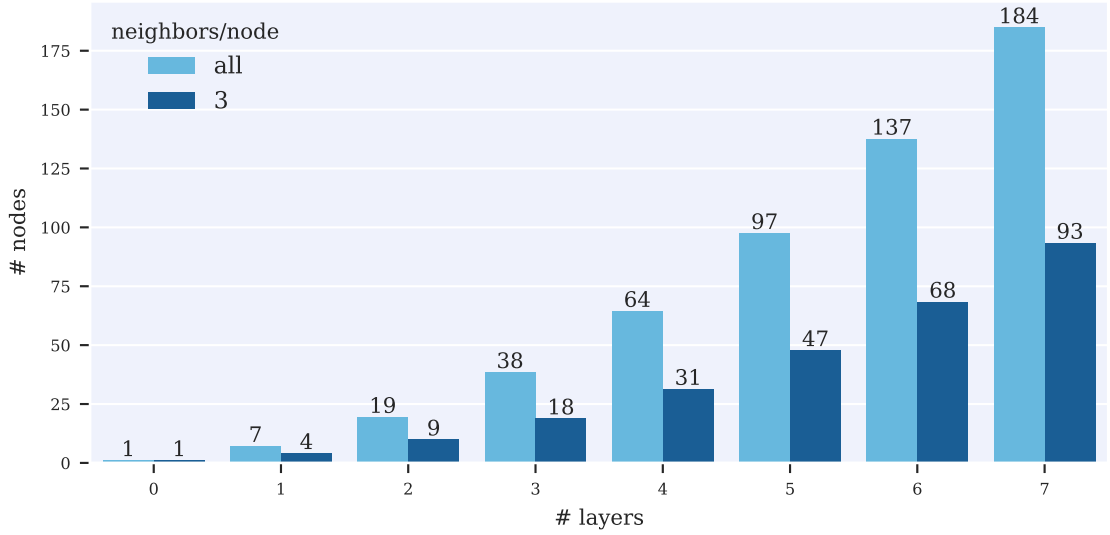



**Figure 7.6:** Average number of nodes used by GNNs to compute output features for a given node. The deeper the model, the more nodes are taken into account, i.e., the more contextual information is available. Stochastic neighborhood sampling reduces the number of nodes taken into account for feature computation. Models without GNN layers (e.g., MLPs) use no neighborhood information.

posed of SAGE and GAT layers. In addition, we examine the performance of MLPs. The comparison to MLPs enables us to assess how the availability of neighborhood information affects performance.

We apply dropout with probability 0.5 to cytoarchitectonic features ($\mathcal{F}_{\mathcal{V}}^{\text{cy}}$) and probabilistic maps ($\mathcal{F}_{\mathcal{V}}^{\text{pm}}$), and add Gaussian white noise with standard deviation 0.1 to the canonical coordinates ($\mathcal{F}_{\mathcal{V}}^{\text{co}}$). All input features are passed through separate fully-connected layers with 512 features, batch normalization, and ReLU activation. If a model uses multiple input features, the results of this projection step are then concatenated. The initial projection step ensures that the model can map the different feature types into a space where concatenation is reasonable.

Depending on the specific model, we then apply a varying number of SAGE, GAT, or fully-connected layers with 512 features, interleaved with batch normalization, ReLU activation, and dropout with probability 0.25. SAGE layers use a neighborhood sampling rate of three (half of the average node degree). GAT layers use dropout with probability 0.5 on the attention coefficients (Equation 2.46). A final linear classification layers maps extracted features to the considered classes (i.e., cytoarchitectonic areas).

The depth and neighborhood sampling strategy of a GNN determine the number of neighborhood nodes that are taken into account for feature computation and classification. Figure 7.6 illustrates how many nodes from the graph are on average

taken into account when computing the output of a given node in one of the cortical midsurface graphs. For example, a GNN with 3 layers and a neighborhood sampling rate of 3 uses 18 neighborhood nodes to create predictions for one node. A model with 7 layers and the same sampling rate takes 93 neighboring nodes into account (on average).

## 7.2 Results

We study the performance of GNNs for cytoarchitecture classification. In line with Chapter 6, we assess the performance by computing the macro F1-score (Equation 2.19) on test and transfer sections that are not included during training. We use the same split into training, test, and transfer sections as for the experiments in Chapter 6, which ensures that results from Chapter 6 are comparable to the results presented in this chapter.

### 7.2.1 Comparison of graph neural network architectures

We compare the performance of different GNN architectures (Section 7.1.4). We use GNNs composed of SAGE and GAT layers. In addition, we compare the performance of GNNs to the performance of MLPs with identical depth, and to the linear evaluation performance of `R50-SCL` from Section 6.2.2. For each layer type, we consider models with three, five, and seven layers. Models investigated in this section only use cytoarchitectonic input features (Section 7.1.1, $\mathcal{F}^{\text{cy}}_{\mathcal{V}_{\text{B}}}$). This makes the results comparable to the results presented in Chapter 6, which also rely solely on cytoarchitectonic features extracted from image patches.

Figure 7.7 shows the test F1-scores of the GNN and MLP models. The linear evaluation performance of `R50-SCL` (Section 6.2.2) is shown for comparison (zero layers).

The results show that GNNs obtain considerably improved classification performance. All GNNs outperform the respective MLP with the same number of layers. MLPs do not obtain improved performance compared to the linear model. This result suggests that simply improving the depth of a model does not improve classification performance. Instead, the improved performance of GNNs can be attributed to the inclusion of 3D topological information provided by the graph structure.

GAT models obtain slightly higher scores than SAGE models. The performance of SAGE models increases moderately with the number of layers. The performance of GAT models is stable and mostly independent from depth. In general, the performance of all considered GNN models is relatively similar. No single model stands out as superior to other GNNs.

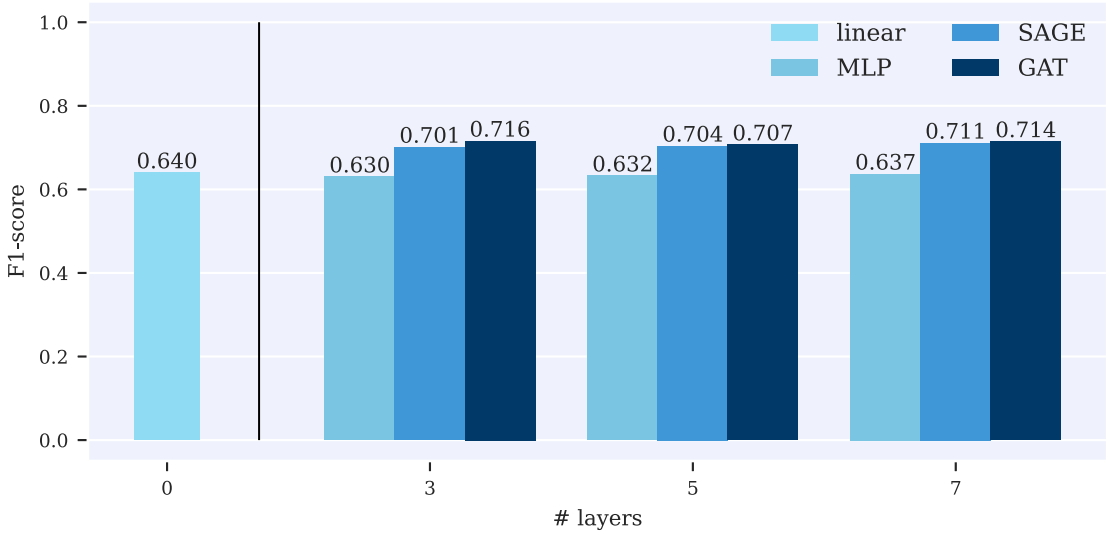



**Figure 7.7:** Test F1-scores for different GNN architectures in comparison to a linear classifier (zero layers) and MLPs. Model are composed of multiple fully-connected layers or GNN layers, followed by a linear layer that maps extracted features to classes. The linear evaluation performance of `R50-SCL` (Section 6.2.2) is given for comparison.

The results show that GNNs obtain considerably improved performance over models that classify each node individually. The investigated GNNs clearly outperform linear models and MLPs with equivalent depth. None of the GNNs models can be identified as being superior to the other GNNs. This suggests that the different GNN models are equally adequate to incorporate 3D context into the classification task. In the following experiments, we use the GAT model with three layers as our default model, denoted as `GAT3-CY` (GAT architecture with 3 layers and cytoarchitectonic features).

### 7.2.2 Influence of node features

The results in Section 7.2.1 demonstrate that GNNs achieve better classification performance than MLP models by combining deep cytoarchitectonic features ($\mathcal{F}_{\mathcal{V}}^{\text{cy}}$) with 3D topological information encoded in the midsurface graphs. In this section, we investigate how the inclusion of additional node features in the form of canonical coordinates ($\mathcal{F}_{\mathcal{V}}^{\text{co}}$) and probabilistic maps ($\mathcal{F}_{\mathcal{V}}^{\text{pm}}$) affects performance. Based on the findings from Section 7.2.1, we use `GAT3-CY` as basis for these experiments.

We conduct experiments using all possible combinations of input features and investigate which features contribute most to performance. In addition, we train a MLP with three layers for each combination of input features. The comparison to

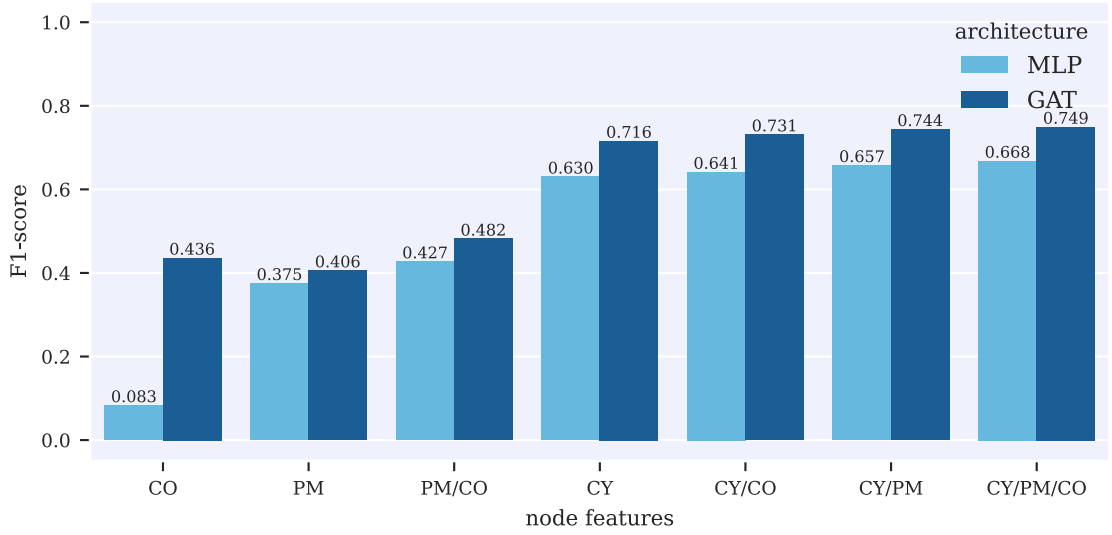



**Figure 7.8:** Test F1-scores obtained by three-layer MLPs and GNNs with GAT architecture and different combinations of node features. Node features include deep cytoarchitectonic features (`CY`, $\mathcal{F}_{\mathcal{V}}^{\text{cy}}$), probabilistic maps (`PM`, $\mathcal{F}_{\mathcal{V}}^{\text{pm}}$), and canonical spatial coordinates (`CO`, $\mathcal{F}_{\mathcal{V}}^{\text{co}}$).

MLPs allows us to assess how the additional features alone (i.e., without topological information from the graph) affect the performance.

Figure 7.8 shows test F1-scores obtained using different combinations of node features. When combined with cytoarchitecture features, the inclusion of canonical spatial coordinates or probabilistic maps improves the test scores. The absolute performance gain resulting from the incorporation of additional node features is comparable between MLPs and GNNs. However, since GNNs achieve higher baseline scores, the overall best performance is obtained by the GNNs.

The inclusion of both canonical spatial coordinates and probabilistic maps improves performance. Incorporating probabilistic maps leads to slightly higher absolute performance. The highest scores are obtained when combining deep cytoarchitectonic features, probabilistic maps, and canonical spatial coordinates. We denote this model as `GAT3-CY/PM/CO` (GAT architecture with 3 layers, cytoarchitectonic features, probabilistic maps, and canonical spatial coordinates).

The performance of models without access to cytoarchitectonic node features is considerably lower. Among these models, the best performance is obtained when using both probabilistic maps and canonical spatial coordinates. The significant performance gap between MLPs and GNNs using only canonical spatial coordinates indicates that the aggregation of features across multiple nodes plays an important role in the efficient use of the provided node features.

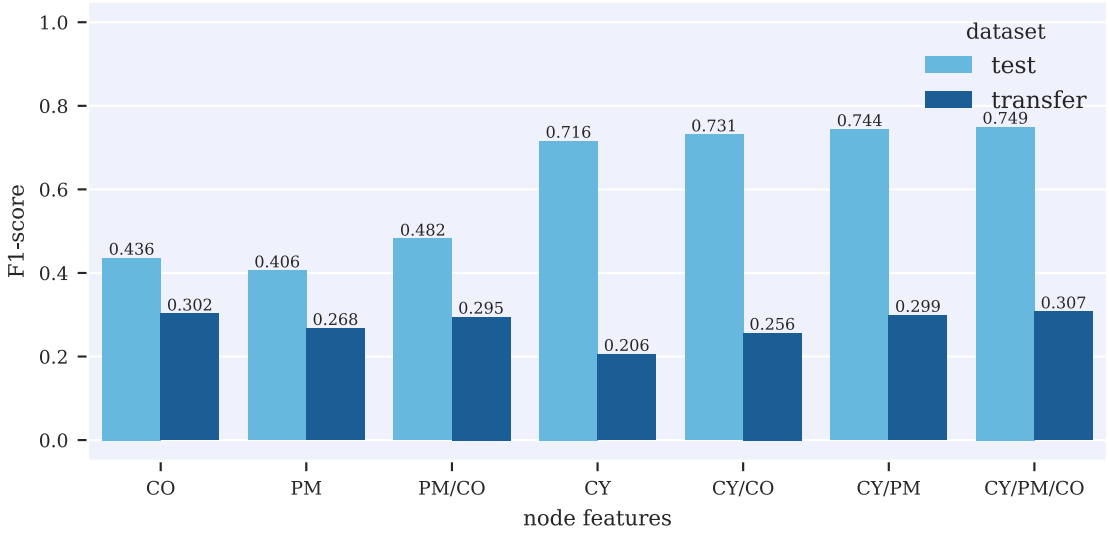



**Figure 7.9:** Test and transfer F1-scores obtained by a three-layer GNN with GAT architecture with different combinations of node features. Node features include deep cytoarchitectonic features (`CY`, $\mathcal{F}_{\mathcal{V}}^{\text{cy}}$), probabilistic maps (`PM`, $\mathcal{F}_{\mathcal{V}}^{\text{pm}}$), and canonical spatial coordinates (`CO`, $\mathcal{F}_{\mathcal{V}}^{\text{co}}$).

Figure 7.9 shows test and transfer F1-scores of three-layer GAT models using different combinations of node features. As in previous comparisons of test and transfer performance, transfer scores turn out to be significantly lower than test scores, indicating suboptimal transferability to unseen brains. Incorporating canonical spatial coordinates or probabilistic maps in addition to cytoarchitectonic features considerably increases the transfer scores. Interestingly, comparable transfer scores can be observed when using no cytoarchitectonic features at all. This result is in line with the previous observation (Section 6.2.3) that learned cytoarchitectonic features are not yet well transferable to unseen brains.

In addition to the above quantitative analysis, we examine the prediction results qualitatively. We visualize the predictions of `GAT3-CY/PM/CO` (the best performing model) on the cortical midsurface graphs of the left hemispheres from brains `B01` and `B04` in Figure 7.10. Both hemispheres are shown from different perspectives, and predictions are shown side-by-side with available annotations for comparison.

The visual comparison shows that the predictions largely align with the available annotations. The shape and position of the predicted cytoarchitectonic areas are generally anatomically plausible. In most cases, the predicted areas are continuous and smooth, which is desired for a microstructural brain parcellation.

The results show that incorporating additional node features containing semantically meaningful information about the brain improves classification performance considerably. Providing additional node information to the GNN models results in

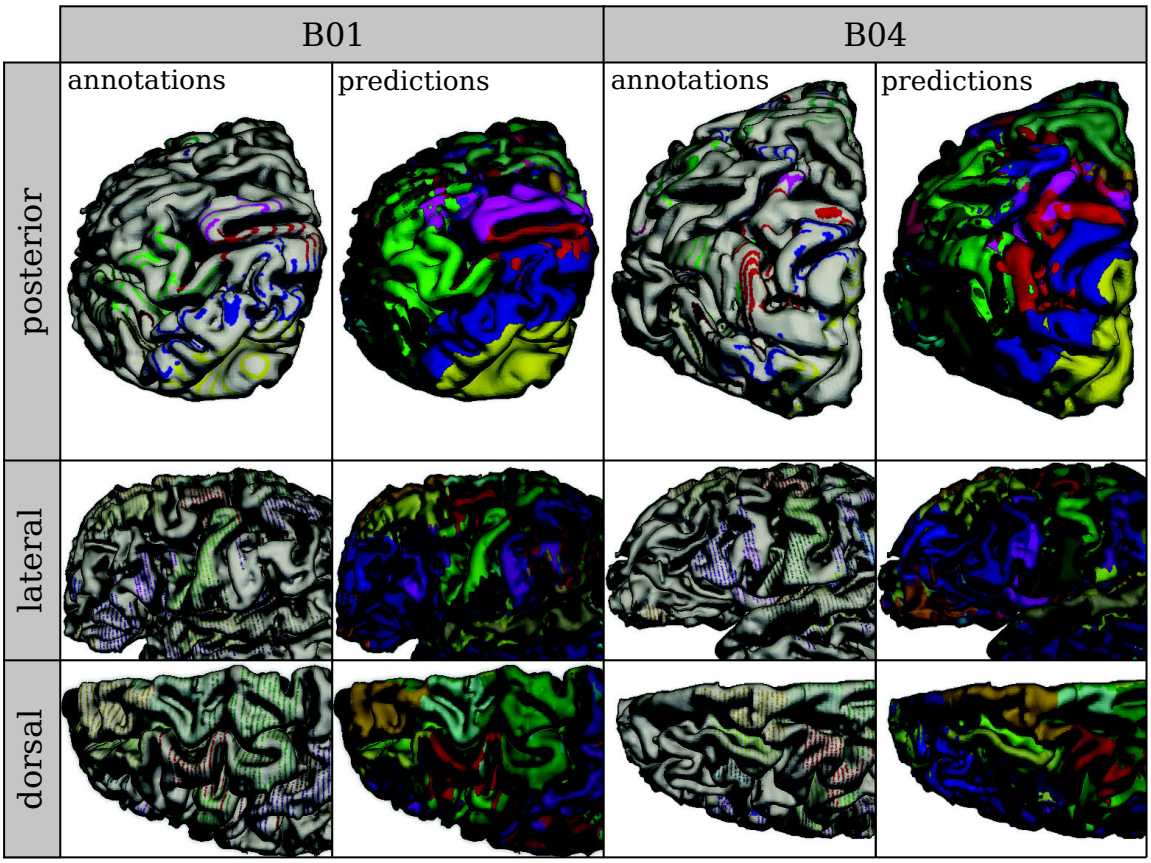



**Figure 7.10:** Predictions for the left hemispheres of brains `B01` and `B04` created by `GAT3-CY/PM/CO`. Results are shown from different perspectives. For each brain, annotations and model predictions are shown side-by-side. Colors encode different brain areas. The visible stripe pattern in the annotations results from the sparse availability of annotations.

the best test classification performance observed so far. The results also confirm previous observations regarding the limited transferability of learned cytoarchitectonic features of new brains.

## 7.3 Summary

In this chapter, we introduced a workflow (Section 7.1.1) to reformulate brain mapping as a node classification task in cortical midsurface graphs and address it using GNNs. The results of our experiments demonstrate that GNNs achieve better performance than models that perform classification based on individual 2D image patches (Section 7.2.1). The comparison of GNNs with MLPs of equal depth suggests that the observed performance gains can be attributed to the incorporation of neighborhood information through the use of GNNs.

Results presented in Section 7.2.2 indicate that probabilistic maps and canonical spatial coordinates complement cytoarchitectonic features and further improve the classification performance. Models that combine cytoarchitectonic features from high-resolution 2D image patches with 3D brain topology, probabilistic atlas information, and canonical spatial coordinates achieve the highest classification scores observed so far.

Sections 8.3 and 8.4 provide comprehensive discussions of the presented results and the role of 3D context for cytoarchitecture classification.

# 8 Discussion

This work addresses deep learning methods for large-scale automated cytoarchitectonic mapping in digitized histological human brain sections. It builds upon previous work (Spitzer et al., 2017, 2018b; Spitzer, 2020) that demonstrated the feasibility of deep learning for classifying cytoarchitectonic areas from the visual system (Section 2.2.7). The work presented in this thesis extends these foundational contributions in two ways:

The method described in Chapter 5 focuses on the practical application of deep learning for supporting cytoarchitectonic mapping by providing an interactive tool for cytoarchitectonic mapping in large series of brain sections. The method integrates well with existing brain mapping workflows and thus allows cytoarchitectonic mapping at the whole-brain level for the first time. The creation of high-resolution 3D cytoarchitectonic maps (Section 5.2.6) is an important example for an analysis workflow enabled by the method. The developed web application ATLaSUI makes the method easily applicable without requiring advanced technical knowledge. Although it requires some user interaction and is not fully automated, it efficiently addresses existing practical challenges. The method thus represents an important contribution to brain mapping.

The methods described in Chapters 6 and 7 address the classification of many cytoarchitectonic areas in different brains. The contrastive learning method presented in Chapter 6 learns to extract cytoarchitectonic features from high-resolution image patches, which provide the basis for cytoarchitecture classification. The GNN method described in Chapter 7 integrates the learned features in a graph-based framework, which efficiently combines cytoarchitectonic features from high-resolution 2D image patches with 3D brain topology. Figure 8.1 summarize the performance of models developed in Chapters 6 and 7. The scores illustrate the continuous improvement in classification performance achieved by the presented methods. They demonstrate the benefit of larger model architectures (`base-CE` vs. `R50-CE`), supervised contrastive learning (`R50-CE` vs. `R50-SCL`), topological information (`R50-SCL` vs. `GAT3-CY`), and prior neuroanatomical knowledge (`GAT3-CY` vs. `GAT3-CY/PM/CO`). The comprehensive evaluations in Chapter 6 further provide valuable insights on label efficiency, transferability, robustness, and semantics of learned features.

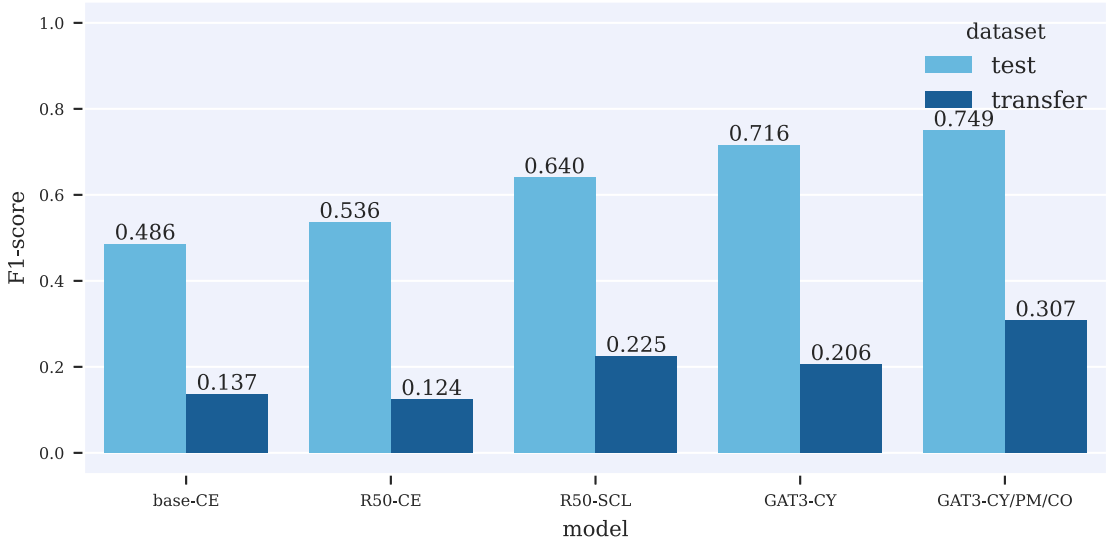



**Figure 8.1:** Test and transfer F1-scores obtained using different deep learning models developed in Chapters 6 and 7. `base-CE` and `R50-CE` are trained using categorical cross-entropy and use `base` and `ResNet50` architectures, respectively (Section 6.2.1). `R50-SCL` is trained using supervised contrastive learning and uses the `ResNet50` architecture. `GAT3-CY` is a three-layer GAT GNN model using deep cytoarchitectonic features learned by `R50-SCL` (Section 7.2.1). `GAT3-CY/PM/CO` is a three-layer GAT GNN model using deep cytoarchitectonic features, probabilistic maps, and canonical spatial coordinates (Section 7.2.2).

Sections 8.1 to 8.6 provide a comprehensive discussion of the presented results and their contribution to the development of automated brain mapping methods. Section 8.7 discusses promising directions for future research.

## 8.1 Accelerating brain mapping with deep learning

Chapter 5 introduces a deep learning method to interactively support and accelerate cytoarchitectonic mapping in large series of brain sections. The method subdivides the overarching classification task into a series of smaller, easier-to-solve subtasks that are addressed by specialized deep neural networks (LSMs). It differs from previously proposed methods for automated cytoarchitecture classification (Spitzer et al., 2017, 2018b) that attempt to train a general model for classifying areas in different brains.

Their locality enables LSMs to exploit the limited variance of individual cytoarchitectonic areas in local brain regions and base predictions on morphological features (e.g., folding patterns) that are not generally representative for cytoarchitecture (Zilles et al., 2012). The distance between annotated training sections can be

adjusted to account for areas with particularly simple or complex properties. This makes the method particularly well suited for interactive application, as it allows users to iteratively provide annotations until a satisfactory segmentation quality is achieved. We demonstrate that reducing the annotation interval improves the segmentation performance for areas hOc3v and hOc5. Similarly, areas with prominent cytoarchitectonic properties (e.g., hOc1) could be segmented using fewer annotations. It is also possible to adapt the annotation intervals for specific regions within an area to improve the local segmentation performance or reduce the annotation effort. This flexibility and the intuitive relationship between annotation effort and segmentation quality contribute to the practical applicability of the method.

The method achieves comparable results for different brains with variable staining, which shows that the method provides sufficient robustness against typical staining variations. No hyperparameter adjustments are required to apply the method to different cytoarchitectonic areas, local regions, and brains. This is an important prerequisite for practical applications, since it does not require users to understand technical aspects of deep learning parameters (e.g., learning rate, training iterations).

The web application ATLaSUI allows using the method without requiring advanced technical knowledge. ATLaSUI has been successfully used in multiple projects conducted at the INM-1, where it considerably reduced the time effort for mapping. ATLaSUI made complete mapping of cytoarchitectonic areas practically feasible for the first time. To illustrate this, we consider the example of area hOc1, which spans 2461 sections in the BigBrain model. Mapping an area on a single section using the GLI-based workflow (Section 2.1.4) takes between 30 and 60 min. Consequently, mapping hOc1 across its full extent would take approximately 150 workdays (assuming 8 h per day). In comparison, mapping hOc1 with the proposed workflow relies only on 18 annotated sections, which can be acquired in about one workday. Including iterative refinement of results and quality control, mapping a large area like hOc1 can be achieved in 1-2 weeks.

We created 3D maps of areas hOc1, hOc2, hOc3v, and hOc5. They represent the first high-resolution models of human cytoarchitectonic areas created from full series of histological sections at cellular resolution. These maps represent an important contribution for studying brain structure at microscopic resolution. The created maps are precise and anatomically consistent. In combination with existing cortical layer maps (Wagstyl et al., 2020) and segmentations of neuronal cells (Upschulte et al., 2022), they enable the systematic analysis of layer- and area-specific organizational principles (Von Economo, 1925).

Although primarily designed for segmentation of cortical areas, the method has already been successfully applied for segmenting several subcortical nuclei, including the six-layered lateral geniculate body (Brandstetter et al., 2021), the medial geniculate body (Kiwitz et al., 2022), the nucleus ventralis intermedius (Devakuruparan

et al., 2021), and several areas in the amygdala. For these studies, neuroanatomists successfully used the developed ATLaSUI without notable support.

The use of multiple specialized LSMs rather than a single general-purpose model provides the basis for the flexibility, improved performance, and practical applicability of the method. However, the decision to use multiple models makes the development and evaluation of the method more challenging. Each model is trained on a different dataset, shows different training behavior, and has to be evaluated on different evaluation data. Changes made to the method (e.g., the evaluation of a new model architecture or a different learning rate) cannot be assessed by training and evaluating a single model, but need to be evaluated for many different models (i.e., different brain areas, local regions, and brains). This process can lead to contradictory results (e.g., when a change improves performance for one area but decreases performance for another area). Thus, the development process is more time-consuming, resource-intensive, and complicated compared to approaches that rely on a single model.

## 8.2 Contrastive learning for large-scale brain mapping

Chapter 6 introduces a contrastive learning method for large-scale cytoarchitecture classification. While the LSM approach uses multiple specialized models to support the mapping of individual areas, this method uses a single general-purpose model that aims to automate the mapping of different areas in multiple brains. Previous work also proposed to use a general-purpose model to classify multiple areas (Spitzer et al., 2017, 2018b). However, these works focus on a few areas of the visual system, while we consider many areas from different parts of the brain.

The proposed method is inspired by the success of self-supervised contrastive learning methods for image classification (Hadsell et al., 2006; Chen et al., 2020; Khosla et al., 2020). However, our experiments (Section 6.2.8) suggest that SimCLR (Chen et al., 2020) and other SSL approaches (Caron et al., 2020; Grill et al., 2020; He et al., 2020; Jain et al., 2020; Khosla et al., 2020; Chen et al., 2021) that base their training objective on data augmentation are not suitable for cytoarchitecture classification.

An underlying assumption of these approaches is that data augmentation preserves relevant features for the downstream task, while varying irrelevant features. While the data augmentation operations we can reasonably use for the given task and data (Section 6.1.3) vary many features that are not relevant for the downstream task (e.g., intensity values or image orientation), some irrelevant features remain unaffected. For example, geometrical and intensity transformations do not change the presence of anatomical landmarks (e.g., blood vessels), the local morphology of the tissue (e.g., folding patterns), or the presence of histological artifacts (e.g.,

damaged tissue). Consequently, identifying differently augmented views of images reduces to identifying transformation-invariant macroscopic features.

This issue could potentially be addressed by reducing the size of image patches, which would prevent the model from recognizing certain macroscopic features (e.g., the folding patterns). However, our experiments (Section 6.2.1) and considerations on cortical thickness (Von Economo, 1925; Zilles et al., 2012) suggest that a sufficiently large image size is necessary to recognize cytoarchitectonic areas. It cannot be expected that the task can be solved with considerably smaller image size.

Another possible approach would be to apply data augmentation operations that vary the appearance or presence of irrelevant macroscopic features. Examples of such transformations include random elastic deformations or masking of specific landmarks. However, it is difficult to ensure that such transformations would not at the same time change relevant cytoarchitectonic features. For example, elastic deformation might alter the relative thickness of cortical layers or the shape of neurons, which are indicators for cytoarchitectonic areas (Section 2.1.4). Given the above considerations, it is questionable if data augmentation-based approaches like SimCLR can be feasibly applied for cytoarchitecture classification.

Our experiments on self-supervised contrastive learning show that approaches that perform well for natural image classification (e.g., photos) are not necessarily effective for cytoarchitecture classification. However, insights gained from these experiments motivated the development of the proposed supervised contrastive learning approach.

The proposed contrastive learning method is supervised, as it relies on the availability of image patches with corresponding annotations. Thus, the requirement for labeled training data does not differ from other commonly used training methods (e.g., categorical cross-entropy). However, in our experiments, models trained with supervised contrastive learning significantly outperform models trained with categorical cross-entropy using the same architecture and amount of training data. The results further indicate that supervised contrastive learning makes more efficient use of available training data (Section 6.2.5). This makes it well suited for cytoarchitecture classification, where creating additional training data is costly. Furthermore, our results suggest that learned features allow classifying areas that are not included during pre-training, making them potentially useful for discovering or classifying new cytoarchitectonic area (Section 6.2.4).

In contrast to categorical cross-entropy, which considers each training sample in isolation (Equation 2.14), the contrastive loss (Equation 6.67) *compares* multiple samples and learns to disentangle them. Thus, the contrastive loss could promote the recognition of subtle cytoarchitectonic features that distinguish different areas. Descriptions of cytoarchitectonic areas in the literature are also often phrased comparatively, i.e., by describing differences to adjacent areas. For example, Rottschy et al. (2007) characterize the cytoarchitecture of area `hOc3v` by describing that cortical

layer II is "distinctly less cell dense in area hOc3d than in hOc3v" and that "layer IV is more cell sparse and has a better defined border to layer V". This "description by comparison" is conceptually similar to principles underlying the supervised contrastive loss.

The feature space analysis (Section 6.2.7) shows that learned features form clusters with anatomically plausible compositions and arrangement in the analyzed t-SNE space. Feature vectors encoding similar areas are located close to each other, and the global structure of the feature space corresponds well with the anatomical organization of the brain.

It should be noted that the training process does not explicitly promote this structure: The contrastive loss optimizes the similarity between feature vectors corresponding to the *same* area. However, there is no mechanism ensuring that different but structurally *similar* areas are also mapped to similar feature representations. The observed correspondence between learned features and cytoarchitectonic organizational principles thus indicates an inherent ability to learn meaningful cytoarchitectonic feature representations. Analyzing the feature space helps to better understand a model's decision process. It further shows that performing clustering within the feature space might be a reasonable method to identify new cytoarchitectonic areas, which would provide a more data-driven way of describing human cytoarchitecture.

Applying trained models to "unseen" brains (i.e., brains that are not included during training) remains a challenge. In our experiments, the effect is observed systematically and independently of the selected transfer brain, indicating that it does not originate from characteristics of a specific brain. This raises the question *why* trained models show systematically decreased performance for unseen brain samples.

The training uses many different samples from different brains with varying appearances. Data augmentation introduces additional variations into the data. The conducted robustness analysis (Section 6.2.6) indicates that models are robust against naturally occurring variations in the data (e.g., tissue staining) and variations resulting from the patch extraction procedure (e.g., translation, rotation). This suggests that the transferability issues likely stem from other sources of variations (e.g., significantly different cytoarchitectonic patterns).

Our experiments suggest that models require brain-specific *and* area-specific training samples to reliably identify an area in a given brain. In particular, models do not seem to derive general brain-specific features from few provided training samples. Simply speaking, a model needs to "see" how a certain brain area "looks like" in a specific brain to reliably recognize and classify it in that brain.

Identifying and addressing the factors underlying the transferability limitations is a crucial step towards automated brain mapping. In its current state, the presented method is applicable to classify areas in brains for which some training data is avail-

able. However, the method cannot reliably classify areas in entirely new brains, which is an important requirement for automated cytoarchitecture analysis at large scale.

The characteristics of the underlying brain-specific differences between cytoarchitectonic areas are still unknown. There is no direct way to identify what kind of features the models use, which is a known shortcoming of deep learning methods (Roscher et al., 2020). Learning more about the features that differentiate different brains thus represents an important direction for future research, both from a methodological and neuroscientific perspective.

## 8.3 Providing 3D context for automated brain mapping

Methods for automated cytoarchitectonic mapping with deep learning, including those presented in this thesis (Chapters 5 and 6) and by Spitzer et al. (2017, 2018b), model brain mapping as classification or segmentation problem on image patches extracted from the cortex. Image patches allow handling of large images and are thus often used when processing entire images is technically infeasible (Ronneberger et al., 2015).

A drawback of patch-wise approaches is their lack of context information, which can be crucial for determining the correct cytoarchitectonic area of an image patch. This becomes evident from the approach neuroanatomists use for brain mapping: Typically, they first roughly determine the location of an area by integrating knowledge about the approximate location of a given section in the brain (e.g., based on the section number or prominent anatomical landmarks) with a mental model of brain anatomy. This step helps to rule out implausible hypotheses without investigating cytoarchitectonic details. For example, a brain section from the frontal part of the brain will never contain visual areas, as these are located in the occipital lobe at the back of the brain.

After pre-localization, the tissue is examined at a higher magnification to analyze cytoarchitectonic properties. Here, neuroanatomists use context information to classify a particular part of the cortex, while patch-wise classification approach are restricted to use relatively small portions of the cortex. This context includes information from adjacent locations in the cortex within the same section or adjacent sections. Including contextual information helps to disambiguate cytoarchitectonic patterns and enables correct classification of cytoarchitectonic areas. In some cases, it further enables classification of obliquely cut tissue (Section 2.1.4) or regions with histological artifacts.

Classification of cytoarchitectonic areas based on isolated image patches is an ill-posed problem. Mimicking existing brain mapping workflows to integrate contextual

information with local texture analysis is a promising approach to improve classification performance. However, while seemingly simple, integrating such contextual information is technically challenging.

Contextual information *within* a brain section could be provided by increasing the size of image patches. This approach is computationally expensive, as the memory and computational requirements grow quadratically with the image patch size. In addition, our results on the limitations of SSL for area classification (Section 6.2.8) suggest that models tend to exploit irrelevant anatomical landmarks, if those landmarks help to solve the task. Models trained on large image patches could also exploit prominent macroscopic landmarks (e.g., cortical folding patterns) to minimize the training loss without learning relevant cytoarchitectonic features.

The incorporation of contextual information across *adjacent* brain sections introduces additional challenges. Extracting a three-dimensional cube (i.e., the three-dimensional equivalent of an image patch) from the brain could provide the necessary context. Deep neural network for segmenting 3D datasets (e.g., MRI or electron microscopy data) use such cubes (Chen et al., 2016a; Milletari et al., 2016), but the lack of accurate 3D brain reconstructions prevents the application of these approaches to histological brain sections. The reconstruction of a brain into a precise 3D model is an elaborate process and cannot yet be routinely performed for many brains (Amunts et al., 2013).

Even when precise 3D reconstruction workflows become available, the technical requirements to process adequately sized cubes are immense. Image patches need to cover the entire depth of the cortex ($\sim 4\,\text{mm}$, e.g., $\sim 2000\,\text{px}$ at $2\,\mu\text{m/px}$) to enable cytoarchitecture classification (Section 6.2.2). A cube of size 2048 in each dimension would require at least 8 GB of memory. We can use the computational requirements in Table 6.1[1] to estimate that processing a single such cube would require 2.5 TB of GPU memory. In light of these estimations, it remains doubtful if the processing of three-dimensional data in this form will become technically feasible in the foreseeable future.

## 8.4 Graph neural networks for brain mapping

Chapter 7 introduces a method for cytoarchitecture classification with GNNs. The method mimics existing brain mapping workflows to integrate contextual information into the classification process. Thus, it addresses the shortcomings of patch-wise image classification discussed in Section 8.3. The formulation as node classification

[1] 2560 GB GPU memory for 2048 image patches with size $2048 \times 2048\,\text{px}$, i.e., approximately 1.25 GB per image patch.

task in an attributed graph makes the method computationally efficient compared to other approaches (Section 8.3).

Formulating brain mapping as node classification task in a graph relies on 3D brain reconstructions. It is important to note that creating a precise 3D brain reconstruction is a complicated process (Amunts et al., 2013) and cannot yet be routinely performed. Thus, the workflow only assumes a coarse and relatively easy-to-compute 3D reconstruction.

The method enables computationally efficient integration of 3D information from attributed cortical midsurface graphs into the classification process. Its ability to classify a location based on neighborhood information (e.g., from adjacent brain sections) and contextual features that complement cytoarchitectonic features (i.e., canonical spatial coordinates, probabilistic atlas information) resembles existing brain mapping workflows. The combination of topological and contextual information through GNNs achieves the best classification performance so far.

The transferability of trained models to unseen brains remains limited. The incorporation of contextual information in addition to cytoarchitectonic features mitigates the effect to some degree, but there remains a considerable performance gap between known and unknown brains. This confirms the limited transferability of learned cytoarchitectonic features to unknown brains. Addressing this shortcoming of the feature learning method (Chapter 6) can be expected to improve the transfer performance of the GNN method.

In the current implementation, node features of input graphs are precomputed and stored on disk for processing. The resulting graphs require relatively little memory (i.e., in the order of a few gigabytes per graph), which makes the method computationally efficient. However, this approach prevents the application of data augmentation to image patches used for cytoarchitectonic feature computation, since all features are precomputed prior to GNN training. Data augmentation would increase the variance in the training data and could potentially improve performance. Computing cytoarchitectonic features from image patches during GNN training would allow data augmentation, at the cost of increased computational requirements for reading and processing image patches.

## 8.5 The role of HPC for automated brain mapping

The size of the used image data and the task's characteristics make automated cytoarchitectonic mapping computationally demanding. Cytoarchitectonic areas are defined by a combination of fine-grained cellular properties (e.g., the size and shape of individual cells) and coarser organizational patterns (e.g., cortical layers). Automated cytoarchitecture classification thus relies on large, high-resolution image

patches, which provide necessary details and a large field of view. This property sets it apart from many other image classification tasks (Russakovsky et al., 2015) and significantly increases the computational requirements.

The encountered computational challenges differ depending on the method: The method introduced in Chapter 5 requires training of hundreds of different models, as each model specializes on a specific brain area in a local region of a single brain. HPC enables parallel training of these many models, which makes it practically feasible.

The supervised contrastive learning method (Chapter 6) relies on fewer models, but uses large training datasets and model architectures. This makes the method computationally expensive (Table 6.1). Despite the availability of HPC systems, several challenges (Section 4.3.3) had to be addressed to meet the requirements of the method.

In comparison, the GNN-based classification approach (Chapter 7) is computationally less demanding, as it relies on precomputed node features. Here, HPC enables the parallel evaluation of multiple models or parameter configurations and thus accelerates the development workflow. As discussed in Section 8.4, the proposed approach could be extended to compute cytoarchitectonic features during GNN training rather than precomputing them, which would allow data augmentation.

Addressing the computational requirements of the developed methods represents an important aspect of this project (Chapter 4). The availability of HPC systems had a huge impact on this work and played a crucial role in enabling most of the research presented in this thesis. Insights gained from overcoming encountered challenges will contribute to the development of future methods. In the future, it can be expected that HPC will become even more important for structural human brain analysis.

## 8.6 Lessons learned for automated brain mapping

The experience gained from this project allows us to formulate several best practices for automated brain mapping with deep learning.

Incorporating domain knowledge into the design of a method is often crucial to apply it successfully. This insight has been a main driver for the development of the presented methods:

- The limited local variability of cytoarchitectonic areas motivated the development of LSMs.
- Literature on cytoarchitecture often describes brain areas by pointing out *differences* to adjacent areas rather than explicitly describing the structure. This inspired the use of contrastive learning, which learns to distinguish brain areas by comparison.

- Existing brain mapping workflows motivated the incorporation of topological and contextual information for cytoarchitecture classification.
- Knowledge of typical cortical thickness and relevant cytoarchitectonic features helped to determine appropriate sizes and resolutions of image patches.

We think that appropriate incorporation of domain knowledge is often more beneficial than using sophisticated general-purpose methods (e.g., powerful deep learning architectures).

The high computational requirements of the presented methods increase the importance of economic and sustainable resources usage (e.g., time and computational resources). The presented methods require significant computational resources (Table 6.1) and are thus associated with considerable economic and ecological costs. It is important to be aware of these costs and use available resources in a well-considered way. For this, we follow several best practices:

- We use optimized software implementations (Chapter 4) to efficiently use available resources (e.g., memory, compute time, file storage). We pay particular attention to computational requirements when adapting existing methods or developing new methods.
- We estimate the potential benefits (e.g., performance gains or new insights) of relevant experiments and prioritize experiments with higher potential impact. In our experience, investigating major methodological changes (e.g., changing from image-based to GNN-based classification) should typically have priority over finetuning potentially less impactful hyperparameters.
- Along these lines, we prefer methods that use few hyperparameters and are robust against hyperparameters changes. For example, Khosla et al. (2020) demonstrate that supervised contrastive learning is more robust to hyperparameter changes than training with categorical cross-entropy. In Chapters 6 and 7, we further use a constant learning rate rather than complex learning rate adaptation schemes (e.g.., stepwise or linear learning rate decay), which reduces the number of hyperparameters.

We think that the described insights will be useful for future work on automated brain mapping and related tasks.

## 8.7 Directions for future research

The results and insights gained from this thesis provide the foundation for future work towards automated cytoarchitectonic mapping at a large scale. The following

describes directions for future work that we believe could contribute most towards this goal.

We described in Chapters 6 and 7 that trained models do not transfer well to unseen brains. Although models are trained with data augmentation (Section 6.1.3) that mimics naturally occurring variations in the data, the results suggest the existence of additional differences between brains. Research on improved transferability might thus not only improve performance, but could also deepen our understanding of inter-individual variability in the human brain (Amunts et al., 2000). Methods from the field of explainable AI (Hendricks et al., 2016; Barredo Arrieta et al., 2020; Roscher et al., 2020) could prove beneficial to better understand how trained models "see" data from different brains. Domain adaptation methods (Tzeng et al., 2014; Gadermayr et al., 2018), which implicitly or explicitly model the shift between related data distributions (e.g., image data from different brains), might also be able to address the observed transferability issues.

We believe that unsupervised or self-supervised feature learning methods represent a promising future research direction, as they can learn semantically meaningful visual features without relying on labeled training data. Although the results presented in Section 6.2.8 indicate that SSL methods based on data augmentation (e.g., SimCLR) are not suitable for brain mapping, it might be possible to develop adequate methods to learn meaningful cytoarchitectonic features without relying on annotations.

Such features have several interesting potential applications: Including microscopic image data from newly acquired brains into the feature learning process could address the transferability issues. In addition, the ability to learn cytoarchitectonic features without annotations could enable data-driven examinations of microstructural human brain organization. A data-driven analysis workflow could encompass the identification of completely new cytoarchitectonic areas through clustering, the analysis of structural gradients, or the correlation of multi-modal measurements with microstructurally defined features. The idea of characterizing microstructure using a set of localized features was already described in 1925 by Constantin Freiherr von Economo in his work on human cytoarchitecture. Von Economo (1925, p. 186) discusses an *ideal brain map*, consisting of well-defined microstructural features (e.g., local cell density or laminar composition) rather than distinct cytoarchitectonic areas defined by subjective human observers. Unsupervised feature learning has the potential to provide such features and could therefore provide the basis for a new generation of cytoarchitectonic human brain mapping.

Although pre-training with the self-supervised distance and location prediction task proposed by Spitzer et al. (2018b) (Section 2.2.7) does not improve classification performance in our experiments (Section 6.2.1), gaining deeper insights into its strengths and limitations might provide a starting point for developing better self-

supervised training objectives. For example, combining this task with approximate 3D reconstructions of multiple brains (Section 7.1.1) might enable self-supervised learning of brain-agnostic features.

The methods presented in this work do not rely on precise 3D brain reconstructions. However, 3D brain reconstructions at cellular resolution are currently under development and have the potential to address some challenges of image-based cytoarchitecture classification (e.g., obliquely cut tissue, Section 4.3.3). Future work should investigate how the availability of precise 3D reconstructed brain volumes could benefit cytoarchitecture classification.

It would be beneficial to quantify the uncertainty of predictions made by trained deep learning models. Uncertainty estimation could point out for which kind of data (e.g., from a specific brain, from specific regions, from specific cytoarchitectonic areas) a model is not confident about its predictions. Such information could prove valuable for method development and the interpretation of results. Examining the applicability of uncertainty estimation (Gal et al., 2016; Kendall et al., 2017) represents an interesting future research direction.

In the cytoarchitecture classification task, we generally have to expect that 1) the data is incompletely annotated (Section 3.3) and that 2) the set of classes is incomplete (i.e., there are cytoarchitectonic areas that are not yet known). These challenges are inherent to the task and should be addressed in future work. Incompletely annotated data (1) could be efficiently incorporated into the training process. For example, the SMILE method (Petit et al., 2018) proposes a heuristic to identify ambiguous annotations and prevent the propagation of incorrect or noisy information during training. This approach could potentially also be applied for cytoarchitecture classification. Approaches that have been successfully used in *open world detection* (Joseph et al., 2021; Mancini et al., 2021) might be used to address the problem of incomplete class sets (2), as they can identify entities from unknown classes (i.e., images from unknown cytoarchitectonic areas) and incrementally learn to identify such entities when annotations become available.

## 8.8 Conclusion

This work represents a step towards automated cytoarchitectonic mapping at a large scale. It presents the first interactive method for accelerated cytoarchitectonic mapping, which enables the analysis of many histological brain sections and the creation of high-resolution 3D cytoarchitectonic maps. Deep neural networks trained with supervised contrastive learning extract meaningful cytoarchitectonic features and accurately predict many cytoarchitectonic areas. The incorporation of topological and contextual information using GNNs mimics existing mapping workflows and im-

proves the classification performance. The results and insights obtained in this work demonstrate the huge potential of deep learning for automated cytoarchitectonic brain mapping and provide valuable foundations for future work.

# A Contributions to publications presented in this thesis

The results and methods presented in this thesis were already partially published in several peer-reviewed publications listed below. For all papers, I, Christian Schiffer, am the first author and was responsible for designing and implementing the described methods, evaluating results, and writing manuscripts. The method for interactive mapping of individual brain areas described in Chapter 5 was conceptualized in discussion with Dr. Hannah Spitzer and Prof. Timo Dickscheid. Publications that were used in this thesis are briefly described below.

### Convolutional Neural Networks for Cytoarchitectonic Brain Mapping at Large Scale (Schiffer et al., 2021f)

**Authors:** Christian Schiffer, Hannah Spitzer, Kai Kiwitz, Nina Unger, Konrad Wagstyl, Alan C. Evans, Stefan Harmeling, Katrin Amunts, Timo Dickscheid

This paper describes the method for interactive mapping of individual brain areas described in Chapter 5, which exploits the limited local variability of individual brain areas to achieve improved classification performance. The model architectures (Section 5.1.5), training procedure (Section 5.1.2), and conducted experiments (Section 5.2) largely follow Schiffer et al. (2021f). The results presented in this thesis were obtained with slightly different hyperparameters (e.g., different learning rates and data augmentation), resulting in different scores. Schiffer et al. (2021f) used an older version of the ATLaS software based on *TensorFlow* (Abadi et al., 2016), while the experiments presented in this thesis use a newer and more efficient version based on *PyTorch* (Paszke et al., 2019). The 3D cytoarchitectonic maps described in Section 5.2.6 use the segmentation results from Schiffer et al. (2021f), so the estimations of surface area and volume of the reconstructed areas match those presented in Schiffer et al. (2021f).

### Contrastive Representation Learning For Whole Brain Cytoarchitectonic Mapping In Histological Human Brain Sections (Schiffer et al., 2021a)

**Authors:** Christian Schiffer, Katrin Amunts, Stefan Harmeling, Timo Dickscheid

This paper introduces the supervised contrastive learning method described in Chapter 6. This thesis adopts the supervised contrastive learning method (Section 6.1.1) using larger deep learning models and performs more extensive evaluation of the method. In particular, this thesis establishes a stronger baseline based on cross-entropy (Section 6.2.1), evaluates more and larger model architectures (Section 6.2.2), extensively evaluates the performance of the method in different application scenarios (Sections 6.2.3 to 6.2.5), examines learned features (Sections 6.2.6 and 6.2.7), and investigates limitations of unsupervised contrastive learning for the given task (Section 6.2.8). The feature analysis conducted in Section 6.2.7 is similar to that performed in Schiffer et al. (2021a), but provides a more in-depth discussion of learned features.

### 2D Histology Meets 3D Topology: Cytoarchitectonic Brain Mapping with Graph Neural Networks (Schiffer et al., 2021c)

**Authors:** Christian Schiffer, Stefan Harmeling, Katrin Amunts, Timo Dickscheid

This paper describes the GNN-based classification approach described in Chapter 7. The paper introduces a framework to reformulate cytoarchitectonic brain mapping as a graph node classification task and investigates the influence of different GNN architectures and additional contextual features. This thesis adopts the graph construction workflow, with some adjustments to improve the quality of created graphs (e.g., exclusion of subcortical gray matter and cleaning using Poisson reconstruction). Experiments in Chapter 7 use cytoarchitectonic features from Chapter 6, which are more expressive than the ones used in Schiffer et al. (2021c). Consequently, the performance obtained using different architectures and node features differs from Schiffer et al. (2021c).

# Related collaborative publications

The following publications are related to the work presented in this thesis, but did not directly contribute to the here presented methods or results. My contributions are specified for each publication.

### Deep Learning Networks Reflect Cytoarchitectonic Features Used in Brain Mapping (Kiwitz et al., 2020)

**Authors:** Kai Kiwitz, Christian Schiffer, Hannah Spitzer, Timo Dickscheid, Katrin Amunts

This paper analyzes features learned by LSMs (Chapter 5) and studies their resemblance to features used during cytoarchitectonic brain mapping. For this, the paper systematically examines feature maps of two LSMs, which were trained to identify areas `hOc1` and `hOc2` in local regions of the BigBrain dataset. The results confirm that trained models capture relevant cytoarchitectonic features, which provides an important foundation for the practical use of the method. For this paper, I trained the neural network models and computed the feature maps. Furthermore, I implemented and performed the similarity computation between features maps, which enabled identification of similar feature maps. For this, I developed a web application for interactive investigation of feature maps.

### Deep Learning-Supported Cytoarchitectonic Mapping of the Human Lateral Geniculate Body in the BigBrain (Brandstetter et al., 2021)

**Authors:** Andrea Brandstetter, Najoua Bolakhrif, Christian Schiffer, Timo Dickscheid, Hartmut Mohlberg, Katrin Amunts

This paper presents 3D cytoarchitectonic maps of the lateral geniculate body (LGB) in the BigBrain dataset, a subcortical nucleus in the thalamus with a distinct six-layered structure. The presented cytoarchitectonic maps were created using the method described in Chapter 5. The six layers of the LGB were annotated and automatically segmented using the ATLaSUI application described in Section 5.2.7. The 3D cytoarchitectonic maps were then computed using the reconstruction workflow described in Section 5.2.6. For this paper, I provided technical support for the use of ATLaSUI, performed the 3D reconstruction, computed the reported surface areas and volumes for each layer of the LGB, and prepared the dataset for publication. The maps were released as part of the multilevel human brain atlas in EBRAINS (`www.ebrains.eu`, Schiffer et al., 2021b).

Kiwitz et al. (2022) presents 3D cytoarchitectonic maps of the medial geniculate body (MGB), which are also released in EBRAINS (Schiffer et al., 2021d). My

contributions to this publication are comparable to Brandstetter et al. (2021). Other publications that use the method from Chapter 5 to compute high-resolution 3D cytoarchitectonic maps of different areas (e.g., subdivisions of the Amygdala) are currently in preparation.

### Learning to Predict Cutting Angles from Histological Human Brain Sections (Schiffer et al., 2021e)

**Authors:** Christian Schiffer, Luisa Schuhmacher, Katrin Amunts, Timo Dickscheid

This paper proposes a deep learning method for automatic prediction of local cutting angles in the cortex based on histological brain sections. It uses a modified U-Net and is trained on data from the BigBrain model, where local cutting angles can be estimated based on the available 3D reconstruction. As regions with oblique cutting angles cannot be reliably classified (Section 2.1.4), identifying such regions has the potential to inform and benefit the methods presented in this thesis. Experiments presented in the paper were conducted by Luisa Schuhmacher during an internship under my supervision. I supervised the work, computed the required training data, provided advice on technical and methodological questions, and wrote the manuscript.

# B The role of reproducible science in this work

The reproducibility of experiments and results is a crucial aspect of scientific work. "Traditional" natural sciences have established best practices (e.g., using lab protocols) to ensure reproducibility of obtained results. However, comparably young research disciplines like computer and data science have long suffered from a lack of reproducibility. Method descriptions, experiments, and results were rarely accompanied by the used software, source code, or datasets that would enable other researchers to reproduce and validate presented results. This lack of reproducibility made it difficult to verify and fully trust many published results. In recent years, the scientific community has fortunately started to attribute much-needed attention to this issue. More and more conferences and journals have started to ask authors for statements on the availability of source code and datasets, often in the form of standardized reproducibility surveys. The results of these surveys can be incorporated into the review process to encourage reproducible research.

In line with these ongoing developments in the scientific community, the experiments and results presented in this work were obtained with reproducibility in mind. In the following, we describe workflows and best practices that we found to improve the reproducibility of our work, some of which might prove useful for others as well.

We use open-source software for our experiments, with few necessary exceptions (e.g., NCCL). Most notably, all research is conducted on Linux-based operating systems (e.g., Arch Linux, Ubuntu, or CentOS), most software and scripts are implemented using *Python* or *bash*, and the majority of used software libraries (see also Section 4.2) and other programs (e.g., *ParaView* (Ahrens et al., 2005) or *meshlab* (Cignoni et al., 2008)) are open source as well. We found that using open-source software generally had a positive impact on the development process. The ability to read the source code of the involved software components often enabled us to identify the source of encountered errors or issues. In some cases, it also allowed us to customize existing software, for example, to fix unreported errors or extend the software's functionality. The active community around large actively maintained open-source projects (e.g., *PyTorch*, Paszke et al., 2019) made troubleshooting easier. In line with the design philosophy of the Unix ecosystem, we found that using well-established tools with clearly defined functionality improved development speed, interoperability,

extensibility, reproducibility, and robustness of developed workflows. In addition, the deliberate decision to avoid non-free software wherever possible lowers the financial requirements for reproducing experiments, making them more accessible.

One of the most important software tools we use for developing software, tracking experiments, and writing this thesis is the version control system *git*[1]. We found that the continuous use of version control for all projects (even for small scripts or side projects) improved development speed and reproducibility in many ways. For example, git provides a concise (and, if necessary, machine-readable) history of all changes made to the tracked files. If used appropriately (e.g., with small commits and descriptive commit messages), this history enables specific rollbacks to earlier versions to reproduce previous results or allows to find out how certain files in the repository were created. In combination with a central repository (e.g., *GitHub* or *GitLab*), git enables collaboration, sharing, or transfer to other computers. We stored our code in a central *GitLab* instance of the FZJ, which made it easy to share code with colleagues or to publish code that enables the reproduction of scientific publications. In addition, we used git to transfer code between computers used for development (e.g., a personal workstation or a laptop) and systems on which computations are performed (e.g., HPC systems). Compared to the alternative method of manually copying all or specific files between used machines, we found that this more systematic method prevents errors resulting from incorrectly or partially copied files.

While git or other version control systems have been widely adopted for software development, datasets required to reproduce experiments are rarely treated in the same way. Instead, datasets or results are often published in dedicated repositories, without sufficient descriptions on what they contain or how they were created, or not published at all. However, with the rising popularity of data-driven methods like machine learning, considering research datasets as "first-class citizens" becomes crucial to ensure reproducibility.

In this thesis, we use the software *datalad*[2] (Halchenko et al., 2021) to address this issue. Datalad is a distributed data management systems, which (among many other features) enables the handling of large datasets within git repositories, as well as provenance tracking to monitor how datasets or results are created. Using the established functionality of git as backbone, datalad offers functions to track how datasets are created, allowing easy reproducibility or recomputation of results. With datalad, repeating one or multiple workflow steps (e.g., after a parameter change or fixing a bug) can often be achieved using a single command. We found that the consistent use of datalad to track the steps of processing workflows makes it easy to comprehend how specific files are created, which is in many cases superior to a

[1] https://git-scm.com
[2] https://www.datalad.org

traditional textual description of the conducted processing steps. In addition, datalad builds upon *git-annex*[3] to enable the inclusion of large files, which integrates well with the aforementioned provenance tracking capabilities. Using the decentral capabilities of git-annex, datalad makes transferring datasets between machines easy. In general, we found that the use of datalad can significantly improve the reproducibility of workflows while inducing only little additional effort.

As a final note, we want to emphasize the role of automation and scripting for reproducibility. Across all tasks related to the creation of this thesis, including software development, data processing, experimentation, evaluation, and scientific writing (including the writing of this thesis), we found that the automation of even seemingly small processing steps can greatly improve reproducibility and development efficiency. For example, we found it beneficial to store all commands that need to be executed to perform a certain processing step (e.g., to process an image or train a neural network) in a script with a descriptive name. Even if a step involves only a few commands that could be easily typed out in a terminal, we prefer to store the commands in a script and include it in the respective git repository. This way, the steps are properly recorded in the project's history, the script is easy to find, it can include additional remarks (e.g., short descriptions or usage instructions), and it can be easily changed and rerun. We found that the additional effort of creating a script to automate a step usually pays off by reducing the effort for future repetitions and by improving the reproducibility of results. To some degree, this approach also influenced our choice of software, as we generally preferred command line software or software that provides programming interfaces (e.g., *ParaView* or *meshlab*) over software that can only be used via a graphical user interface.

Taking this idea a step further, we often used the workflow management system *snakemake*[4] (Mölder et al., 2021). Snakemake provides a Python-based workflow description language, allowing users to define dependencies between different files, including the necessary steps to create a specific output file from a range of input files. Snakemake makes it easy to trace even complex dependencies between many different files. We used snakemake to create most of the figures presented in this thesis (e.g., by defining processing steps to create performance plots from prediction results).

In conclusion, we found that being aware of reproducibility can greatly improve the quality of a research project's outcome. In many cases, doing so can also improve the efficiency of workflows, be it during software development, experimentation, or writing of scientific reports. Although many aspects can still be improved, we think

[3] https://git-annex.branchable.com
[4] https://snakemake.github.io

that following the described practices has contributed to the reproducibility of our work and consequently to the quality of this thesis.

# Glossary

## Mathematical notation

| | |
|---|---|
| $\mathbb{R}$ | Real numbers |
| $\mathbb{R}^{\geq 0}$ | Real numbers greater or equal to zero |
| $\mathbb{N}$ | Natural numbers |
| $x$ | A scalar value |
| $\boldsymbol{x}$ | A column vector |
| $\boldsymbol{x}(i)$ | Element $i$ of vector $\boldsymbol{x}$ |
| $M$ | A matrix, image, or higher-dimensional tensor |
| $I(i, j)$ | Element at row $i$ and column $j$ in image $I$ (analogue for higher-dimensional tensors) |
| $*$ | Convolution |
| $\nabla_\theta$ | Nabla operator (partial derivatives with respect to parameter vector $\theta$) |
| $\|\|$ | Concatenation operator |
| $\|$ | Divisible operator ($a \mid b \Leftrightarrow b \mod a = 0$) |
| $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ | Scalar product of vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ |
| $\|\boldsymbol{x}\|_2$ | Euclidean norm of vector $\boldsymbol{x}$ |
| $\mathcal{A} = \{1, 2\}$ | Set containing elements 1 and 2 |
| $f(x; \theta)$ | Function with argument $x$, parameterized by $\theta$ |
| $f \circ g$ | Composition of functions $f$ and $g$ |
| $\mathbb{I}_A$ | Indicator function (1 if condition $A$ is true, 0 otherwise) |
| $U[a, b]$ | Uniform distribution over interval $[a, b]$ |
| $\mathbb{E}_{\mathcal{X}}[x]$ | Expected value of $x$ with respect to a distribution $\mathcal{X}$ |

## Symbols & functions

| | |
|---|---|
| $f(\boldsymbol{x}; \theta)$ | Deep neural network with input $\boldsymbol{x}$ and parameters $\theta$. 19 |
| $\theta \in \Theta$ | Parameters of a deep neural network from a parameter space $\Theta$. 19 |

| | |
|---|---|
| $\boldsymbol{x}$ | Input vector for a deep neural network or a layer. 19 |
| $l(\hat{\boldsymbol{y}}, \boldsymbol{y})$ | Loss function measuring the disagreement between a prediction $\hat{\boldsymbol{y}}$ and the expected groundtruth $\boldsymbol{y}$. 19 |
| $\hat{\boldsymbol{y}}$ | Prediction of a deep neural network. 19 |
| $\boldsymbol{y}$ | Target value for a prediction (also referred to as groundtruth or labels). 19 |
| $\mathbb{X}$ | Training dataset sampled from a data distribution $\mathcal{X}$. 19 |
| $n$ | Number of training examples in the training dataset $\mathbb{X}$. 19 |
| $L(\mathbb{X}; \theta)$ | Loss function measuring the prediction performance of a deep neural network parameterized by $\theta$ with respect to a dataset $\mathbb{X}$. 19 |
| $\Delta\theta^{(i+1)}$ | Parameter update in iteration $i$ of gradient descent optimization. 20 |
| $\lambda$ | Learning rate for training with gradient descent or SGD. 20 |
| $\mathbb{B}$ | Batch of $b$ training samples used by SGD. 21 |
| $b$ | Number of training samples in a batch $\mathbb{B}$. 21 |
| $\mu$ | Momentum factor for SGD with momentum. 22 |
| $\omega$ | Weight decay factor for optimization with L2 regularization. 23 |
| $\eta$ | Trust factor used by the LARS optimizer. 23 |
| $c$ | Number of classes for classification or segmentation tasks. 24 |
| softmax | Softmax function. 24 |
| $l_{CE}$ | Categorical cross-entropy loss function. 25 |
| $\mathbb{X}^{te}$ | Test dataset for performance evaluation. 25 |
| CM | Confusion matrix. Entry at row $i$ and column $j$ specifies how many samples belonging to class $i$ are classified as class $j$. 26 |
| accuracy | Accuracy. Fraction of correctly classified samples. 26 |
| $\text{precision}_k$ | Precision for class $k$. Measures how likely a sample belongs to class $k$ if a model predicts it belongs to class $k$. 26 |
| $\text{recall}_k$ | Recall for class $k$. Measures how likely a sample predicted to belong to class $k$ actually belongs to class $k$. 26 |
| $\text{F1}_k$ | F1-score for class $k$, computed as harmonic mean between precision and recall. 27 |
| F1 | Macro F1-score, computed as average of class-specific F1-scores. 27 |

| | |
|---|---|
| $d_i$ | Input dimension of a neural network layer. 28 |
| $\mathrm{FC}(\boldsymbol{x}; W, \boldsymbol{b})$ | Fully-connected layer with input $\boldsymbol{x}$, weight matrix $W$, and bias vector $\boldsymbol{b}$. 28 |
| $W \in \mathbb{R}^{d_o \times d_i}$ | Weight matrix of a fully-connected layer with input dimension $d_i$ and output dimension $d_o$. 28 |
| $\boldsymbol{b} \in \mathbb{R}^{d_o}$ | Bias vector of a fully-connected layer with output dimension $d_o$. 28 |
| $d_o$ | Output dimension of a neural network layer (also referred to as *number of features*). 28 |
| $\sigma$ | Non-linear activation function. 28 |
| sigmoid | Sigmoid activation function. 29 |
| ReLU | ReLU activation function. 30 |
| LeakyReLU | Leaky ReLU activation function. 30 |
| $\mathrm{BN}(\boldsymbol{x}; \boldsymbol{\gamma}, \boldsymbol{\beta})$ | Batch normalization with input $\boldsymbol{x}$ and parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$. 30 |
| $\boldsymbol{\mu}_{BN}$ | Mean for batch normalization. 30 |
| $\boldsymbol{\sigma}_{BN}$ | Standard deviation for batch normalization. 30 |
| $\mathrm{drop}_i(\boldsymbol{x}; p)$ | Dropout layer. Sets entry $i$ of input vector $\boldsymbol{x}$ to zero with probability $p$. 31 |
| $X \in \mathbb{R}^{h \times w}$ | Input image for a deep neural network or a layer with height $h$ (number of rows) and width $w$ (number of columns). 32 |
| $Y$ | Output image of a deep neural network layer. 33 |
| $n_K$ | Number of filters of a convolutional layer. 33 |
| $K$ | Set of $n_K$ filters of a convolutional layer. 33 |
| $K_k \in \mathbb{R}^{u \times u \times c}$ | Filter of a convolutional layer with size $u$ and $c$ input channels. 33 |
| $\mathrm{conv}(X; K, \boldsymbol{b})$ | Convolutional layer with input image $X$, filters $K$, and bias vector $\boldsymbol{b}$. 36 |
| $\mathrm{pool}(X)$ | Pooling layer with input image $X$. 37 |
| $\mathcal{G}$ | Graph with nodes $\mathcal{V}$, edges $\mathcal{E}$, and node features $\mathcal{F}_\mathcal{V}$. 37 |
| $\mathcal{V}$ | Nodes of a graph. 37 |
| $v_i$ | A node in a graph. 37 |
| $n_\mathcal{V}$ | Number of nodes in a graph 37 |
| $\mathcal{E}$ | Edges of a graph. 37 |
| $\mathcal{N}(i)$ | Neighborhood of a node $v_i$. 37 |
| $\mathcal{N}_k(i)$ | $k$-hop neighborhood of a node $v_i$. 38 |
| $A$ | Adjacency matrix of a graph. 38 |
| $\mathcal{F}_\mathcal{V}$ | Node features of a graph. 38 |

| | |
|---|---|
| $\text{mp}\big(\boldsymbol{x}_{v_i}; \theta_{msg}, \theta_{upd}\big)$ | Message passing function with input node features $\boldsymbol{x}_{v_i}$ and parameters $\theta_{msg}$ and $\theta_{upd}$. 39 |
| $\text{msg}\big(\boldsymbol{x}_{v_i}; \theta_{msg}\big)$ | Message function of the GNN message passing framework with input node features $\boldsymbol{x}_{v_i}$ and parameters $\theta_{msg}$. 39 |
| agg | Aggregation function of the GNN message passing framework. 39 |
| upd | Update function of the GNN message passing framework. 39 |
| $\mathcal{G}^{(k)}$ | Graph produced by $k$ message passing layers. 39 |
| $\mathcal{F}_{\mathcal{V}}^{(k)}$ | Node features of a graph produced by $k$ message passing layers. 39 |
| $\mathbb{B}_{\mathcal{V}}$ | Batch of nodes from a graph. 40 |
| $\mathcal{G}_{\mathbb{B}}$ | Subgraph created from a batch of nodes $\mathbb{B}_{\mathcal{V}}$. 40 |
| $\mathcal{F}_{\mathcal{V}_{\mathbb{B}}}$ | Node features of subgraph $\mathcal{G}_{\mathbb{B}}$. 40 |
| $\mathcal{V}_{\mathbb{B}}$ | Nodes of subgraph $\mathcal{G}_{\mathbb{B}}$. 40 |
| $\mathcal{E}_{\mathbb{B}}$ | Edges of subgraph $\mathcal{G}_{\mathbb{B}}$. 40 |
| $\text{agg}_{\text{SAGE}}\big(\boldsymbol{x}_{v_i}\big)$ | Aggregation function of a SAGE layer with input node features $\boldsymbol{x}_{v_i}$. 41 |
| $\text{upd}_{\text{SAGE}}\big(\boldsymbol{x}_{v_i}; W, \boldsymbol{b}\big)$ | Update function of a SAGE layer with input node features $\boldsymbol{x}_{v_i}$ and parameters $W \in \mathbb{R}^{d_o \times 2d_i}$ and $\boldsymbol{b} \in \mathbb{R}^{d_o}$. 41 |
| $\text{agg}_{\text{GAT}}\big(\boldsymbol{x}_{v_i}\big)$ | Aggregation function of a GAT layer with input node features $\boldsymbol{x}_{v_i}$. 42 |
| $\text{upd}_{\text{GAT}}\big(\boldsymbol{x}_{v_i}; W, \boldsymbol{b}\big)$ | Update function of a GAT layer with input node features $\boldsymbol{x}_{v_i}$ and parameters $W \in \mathbb{R}^{d_o \times 2d_i}$ and $\boldsymbol{b} \in \mathbb{R}^{d_o}$. 42 |
| $\tilde{X}_i$ | View of an image $X_i$. 51 |
| $j(i)$ | Index of corresponding other view in a batch $\tilde{\mathbb{B}}$ for SimCLR training. 51 |
| $\tilde{\mathbb{B}}$ | Batch with $2b$ image views for SimCLR training. 52 |
| $L^{\text{ssl}}\Big(\tilde{\mathbb{B}}\Big)$ | Contrastive loss function of batch $\tilde{\mathbb{B}}$ for SimCLR training. 52 |
| $l^{\text{ssl}}(i)$ | Contrastive loss function of sample $i$ in $\tilde{\mathbb{B}}$ for SimCLR training. 52 |
| $\boldsymbol{z}_i \in \mathbb{R}^{d_z}$ | Projected feature vector used in contrastive learning. 52 |
| $d_z$ | Dimension of projected feature vector $\boldsymbol{z}_i$. 52 |
| $f_E$ | Encoder for contrastive learning. 52 |
| $\boldsymbol{h}_i \in \mathbb{R}^{d_h}$ | Feature vector produced by contrastive learning encoder $f_E$. 52 |

| | |
|---|---|
| $d_h$ | Dimension of feature vector $\boldsymbol{h}_i$. 52 |
| $f_P$ | Projection head for contrastive learning. 52 |
| $\mathrm{sim}(\boldsymbol{u}, \boldsymbol{v})$ | Cosine similarity between vectors $\boldsymbol{u}$ and $\boldsymbol{v}$. 52 |
| $\tau$ | Temperature parameter for NT-Xent loss function. 52 |
| $f_C$ | Linear classifier for linear evaluation after contrastive learning. 53 |
| $L^{\mathrm{ssl}+}\left(\tilde{\mathbb{B}}\right)$ | Contrastive loss function of batch $\tilde{\mathbb{B}}$ for supervised contrastive learning (Khosla et al., 2020). 54 |
| $l^{\mathrm{ssl}+}(i)$ | Contrastive loss function of sample $i$ in $\tilde{\mathbb{B}}$ for supervised contrastive learning (Khosla et al., 2020). 54 |
| $c_i$ | Class of image at batch index $i$. 54 |
| $n_{c_i}$ | Number of classes in a batch belonging to $c_i$. 54 |
| $\mathtt{B}$ | Identifier of a brain (e.g., $\mathtt{B01}$). 60 |
| $S_s^{\mathtt{B}}$ | Microscopic image of a histological brain section with section number $s$ from brain $\mathtt{B}$. 60 |
| $\Phi_{s \to t}$ | Linear transformation aligning a section $s$ to a section $t$. 64 |
| $P_s^{\mathtt{B}}$ | Probabilistic map projected onto section $s$ of brain $\mathtt{B}$. 66 |
| $R_s^{\mathtt{B}}$ | Canonical spatial coordinates from MNI-Colin27 space projected onto section $s$ of brain $\mathtt{B}$. 67 |
| $\mathcal{A}$ | Set of 113 considered cytoarchitectonic areas (Table 3.2). 68 |
| $\mathtt{A}$ | Placeholder for a cytoarchitectonic area from $\mathcal{A}$. 68 |
| $\mathrm{LSM}_{\mathtt{B}}^{s \overset{\mathtt{A}}{\leftrightarrow} t}$ | LSM for segmenting area $\mathtt{A}$, trained on a pair of sections $s$ and $t$ from brain $\mathtt{B}$. 82 |
| $\mathcal{R}_s^{\mathtt{B}}(\mathtt{A}; r)$ | ROI for LSM training with radius $r$ around area $\mathtt{A}$ on section $s$ from brain $\mathtt{B}$. 84 |
| $\tilde{\mathcal{R}}_u^{\mathtt{B}}(\mathtt{A}; r, s, t)$ | Estimated ROI for LSM inference on section $u$ from brain $\mathtt{B}$. 84 |
| $L^{\mathrm{sup}}(\mathbb{B})$ | Supervised contrastive loss function of batch $\mathbb{B}$. 108 |
| $l^{\mathrm{sup}}(i)$ | Supervised contrastive loss function of sample $i$ in $\mathbb{B}$. 108 |
| $\mathcal{B}$ | Brains used in Chapters 6 and 7. 109 |
| $B_h$ | Brain for transferability evaluation (holdout brain). 109 |
| $\mathcal{B}_{tt}$ | Brains for training and testing. 109 |
| $\mathcal{S}_{tt}$ | Brain sections for training and testing. 109 |
| $\mathcal{S}_{tr}$ | Brain sections for training. 109 |
| $\mathcal{S}_{te}$ | Brain sections for testing. 109 |
| $\mathcal{S}_h$ | Brain sections for transferability evaluation. 109 |
| $n_s$ | Number of sampled image patches per area (*sampling rate*). 111 |

| | |
|---|---|
| $r_{\mathtt{A}}$ | Ratio between the sampling rate $n_s$ and total number of samples $n_{\mathtt{A}}^t$ available for an area `A`. 111 |
| $n_{\mathtt{A}}^t$ | Total number of samples available for an area `A`. 111 |
| $\bar{r}_{\mathcal{A}}$ | Median sampling ratio across all areas in $\mathcal{A}$. 111 |
| $\mathcal{G}_{\mathtt{B}}$ | Graph representing the cortical midsurface of brain `B`. 139 |
| $\mathcal{V}_{\mathtt{B}}$ | Nodes of graph $\mathcal{G}_{\mathtt{B}}$. 139 |
| $\mathcal{E}_{\mathtt{B}}$ | Edges of graph $\mathcal{G}_{\mathtt{B}}$. 139 |
| $\mathcal{F}_{\mathcal{V}_{\mathtt{B}}}^{\mathrm{cy}}$ | Cytoarchitectonic nodes features of graph $\mathcal{G}_{\mathtt{B}}$. 143 |
| $\mathcal{F}_{\mathcal{V}_{\mathtt{B}}}^{\mathrm{co}}$ | Canonical spatial coordinates for nodes of graph $\mathcal{G}_{\mathtt{B}}$. 143 |
| $\mathcal{F}_{\mathcal{V}_{\mathtt{B}}}^{\mathrm{pm}}$ | Probabilistic node features of graph $\mathcal{G}_{\mathtt{B}}$. 144 |
| $\mathcal{F}_{\mathcal{V}_{\mathtt{B}}}^{\boldsymbol{y}}$ | Annotations for nodes of graph $\mathcal{G}_{\mathtt{B}}$. 144 |

# Abbreviations

**AAHA** Allen Adult Human Brain Atlas

**API** application programming interface

**BDA** Big Data Analytics

**FZJ** Forschungszentrum Jülich

**GPU** graphics processing unit

**GSM** global segmentation model

**HHU** Heinrich-Heine-University Düsseldorf

**HPC** high-performance computing

**INM-1** Institute of Neuroscience and Medicine

**JSC** Jülich Supercomputing Centre

**LSM** local segmentation model

**MNI** Montréal Neurological Institute, McGill University

**ROI** region of interest

**SSH** secure shell

# Terms & definitions

**allocortex**

Cortical regions with variable laminar patterns (see also isocortex). 6

**automatic mixed precision (AMP)**

Memory and runtime efficient training of deep neural networks based on half-precision computations. 114

**anterior**

Anatomical direction: Towards the front (see Figure 2.1, right). 6

`base`

CNN architecture resembling the encoder of the U-Net for cytoarchitecture segmentation proposed in Spitzer et al. (2017). 115

**architecture**

Layer composition and connectivity of a deep neural network. 19

**ATLaS**

Software framework for large-scale analysis of microscopic images. 72

**ATLaSUI**

Web-based user interface for the ATLaS framework. 102, 103

**receptor autoradiography**

Imaging technique to visualize receptor densities in the brain. 7

**backpropagation**

Algorithm for gradient computation during deep neural network training. 18

`base-CE`

`base` model for large-scale cytoarchitecture classification trained with categorical cross-entropy loss. 154

**BigBrain**

High-resolution 3D human brain atlas reconstructed from 7404 histological brain sections (Amunts et al., 2013). viii, 10

**batch normalization**

Normalization strategy for deep neural network training (Ioffe et al., 2015). 30

**brainstem**

Part of the human brain connecting the cerebrum to the spinal cord. 5

**categorical cross-entropy**

Loss function for multi-class classification tasks. 24

**cerebellum**

Heavily folded lower part of the brain containing the majority of nerve cells in the brain. 5

**cerebrum**

Largest part of the human brain, consisting of two hemispheres with a highly convoluted surface. 6

**convolutional neural network (CNN)**

Deep neural network architecture for processing of image data. 18

**contrastive learning**

Framework for feature learning from pairwise comparison of similar and dissimilar data points. 2

**corpus callosum**

Bundle of nerve fibers connecting the two hemispheres of the cerebrum. 5

**cortex**

Outer layer of the cerebrum with a high density of neuronal cells. 6

**cortical layer**

Layers within the cortex. 6

**computed tomography (CT)**

Radiation-based imaging technique enabling in-vivo structural measurements inside the body. 7

**cytoarchitectonic area**

Brain region with distinct cytoarchitectonic properties. 2

**cytoarchitecture**

Composition of neurons in the cortex, including cell distribution, size, orientation, and presence of certain cell types. 2

**distributed deep learning (DDL)**

Techniques for distributed training of deep neural networks across multiple GPUs. 55

**deep learning**

Machine learning technique for analyzing complex high-dimensional data (e.g., images). 1

**deep neural network**

Differentiable and parameterized function composed of smaller functional modules (layers). 19

**densely connected networks (DenseNet)**

Deep neural network architecture for image analysis using densely connected layers. 32, 46

**depth**

Number of layers in a deep neural network. 19

**diffusion weighted MRI (DWI)**

Imaging technique that measures the diffusion of water molecules in the body. 10

**ex-vivo**

Outside of a living organism. 7

**fully-connected layer**

General purpose neural network layer composed of a linear transformation and an activation function. 28

**functional magnetic resonance imaging (fMRI)**

Imaging technique enabling in-vivo measurements of functional activity in the brain. 7

**fusiform cell**

Non-pyramidal cells with variable (polymorph) appearance. 12

**global average pooling (GAP)**

Pooling operation that reduces features into a vector. 37

**graph attention network (GAT)**

GNN architecture using attention mechanisms for weighting node features (Veličković et al., 2018). 41, 42

`GAT3-CY`

Three-layer GAT model using deep cytoarchitectonic features produced by `R50-SCL`. 148

`GAT3-CY/PM/CO`

Three-layer GAT model using deep cytoarchitectonic features produced by `R50-SCL`, probabilistic maps, and canonical spatial coordinates. 149

**graph convolutional network (GCN)**

Generalization of CNNs for graph structured data. 38

**gradient descent**

Iterative gradient-based optimization algorithm. 20

**gray level index (GLI)**

Intermediate image representation used in Schleicher et al. (1999) to measure the volume fraction of neurons. 16, 17

**graph neural network (GNN)**

Deep neural network architecture for processing of graph structured data. 38

**gradient checkpointing**

Strategy for memory efficient training of deep neural networks using gradient recalculation (Chen et al., 2016b). 56

**General Parallel File System (GPFS)**

Distributed file system by IBM. Used by JUST. 72

**gray matter**

Brain tissue with high neuron density (see cortex, subcortical gray matter). 5

**gray-white matter boundary**

Boundary between cortex and white matter. 6

**gyrification**

Folding pattern of the brain surface. 5

**gyrus**

Outward fold of the brain surface. 6

**HPST (High Performance Storage Tier)**

High-performance storage layer of JUST based on flash storage. 72

**high-resolution U-Net (HR U-Net)**

U-Net architecture for cytoarchitecture segmentation based on high-resolution image features. 87, 88

**immunohistochemistry**

Imaging technique to visualize different cell types. 7

**in-vivo**

Inside of a living organism. 7

**inferior**

Anatomical direction: Towards the bottom (see Figure 2.1, right). 6

**isocortex**

Cortical regions with a regular six-layered organization. 6

**Julich-Brain**

Probabilistic human brain atlas created from superimposed mappings of cytoarchitectonic areas from 23 brains (Amunts et al., 2020). 9

**JURECA**

Supercomputer system at JSC (Krause et al., 2018). Decommissioned at the end of 2020. 71

**JURECA-DC**

Supercomputer system at JSC (Krause et al., 2018). Successor of JURECA. 71

**JUST**

Storage cluster at JSC. 71

**layer-wise adaptive rate scaling (LARS)**

Variant of SGD for training with large batch sizes. 23

**lateral**

Anatomical direction: Towards the side/sideways (see Figure 2.1, right). 6

**layer**

Functional building block used to construct deep neural networks. 19

**lobe**

Anatomical subdivision of the brain based on anatomical landmarks. 6

**loss function**

Function measuring the difference between predictions of a model and a given target value. 19

**low-resolution U-Net (LR U-Net)**

U-Net architecture for cytoarchitecture segmentation based on low-resolution image features. 88, 89

**medial**

Anatomical direction: Towards the center (see Figure 2.1, right). 6

**Microdraw**

Web-application for visualization and annotation of high-resolution image data. 102

**multi-layer perceptron (MLP)**

General purpose neural network architecture composed of multiple fully-connected layers. 29

**MNI-Colin27**

Single-subject reference space derived from MRI measurements (Holmes et al., 1998). 65

**Message Passing Interface (MPI)**

Software standard for communication between processes in parallel applications. 73

**magnetic resonance imaging (MRI)**

Imaging technique enabling in-vivo measurements of tissue structures. 7

**multi-scale U-Net (MS U-Net)**

U-Net architecture for cytoarchitecture classification combining low- and high-resolution image features. 87–89

**NVIDIA Collective Communication Library (NCCL)**

Software library for collective communication between GPUs. 74

**nerve fiber**

Fibers connecting neurons to enable transmission of chemical and electrical signals. 5

**neuron**

Nerve cell. 5

**normalized temperature-scaled cross-entropy (NT-Xent)**

Temperature-scaled variant of cross-entropy for contrastive learning. 52

**nucleus**

Structurally distinct cluster of neurons in the subcortical gray matter. 5

**principal component analysis (PCA)**

Dimensionality reduction technique based on eigenvector projection. 47

**pial boundary**

Boundary between cortex and the outside of the brain. 6

**polarized light imaging (PLI)**

Microscopic imaging technique enabling the measurement of nerve fiber directions in postmortem brain tissue. 7

**posterior**

Anatomical direction: Towards the back (see Figure 2.1, right). 6

**granular cell**

Round-shaped cell. 12

**pyramidal cell**

Approximately triangular shaped neuronal cell. 12

**RAS coordinate system**

Coordinate convention for anatomical data. Coordinate axes correspond to left to right, posterior to anterior, and inferior to superior directions, respectively. 67

**rectified linear unit (ReLU)**

Truncated linear activation function used in deep neural networks. 29, 30

`R50-CE`

`ResNet50` model for cytoarchitecture classification trained with categorical cross-entropy loss. 118

**residual network (ResNet)**

Deep neural network architecture for image analysis using residual connections. 32, 44

`R50-SCL`

`ResNet50` model for cytoarchitecture classification trained with supervised contrastive learning. 119

**SAGE**

GCN layer that uses sampling and aggregation for learning on graphs (Hamilton et al., 2017). 41

**stochastic gradient descent**

Computationally efficient variant of gradient descent that operates on batches of data. 21

**SimCLR**

Self-supervised contrastive learning method that defines image similarity based on data augmentation (Section 2.2.5, Chen et al., 2020). 51

**softmax function**

Activation function that normalizes a vector. 24

**self-supervised learning (SSL)**

Training paradigm for feature learning from auxiliary tasks. 48

**subcortical gray matter**

Regions of the gray matter inside of the cerebrum. 5

**sulcus**

Inward fold of the brain surface. 6

**superior**

Anatomical direction: Towards the top (see Figure 2.1, right). 6

**Hierarchical Data Format 5 (HDF5)**

Hierarchical data format for storing array-like data (The HDF Group, 1997). 79

**Tagged Image File Format (TIFF)**

File format for storing image data. 76

**training**

In the context of deep learning, training describes the process of optimizing the parameters of a deep neural network to make it solve a specific task. 19

**U-Net**

Deep neural network architecture for image segmentation (Ronneberger et al., 2015). 32

**weight decay**

Regularization technique applied to prevent overfitting in deep neural networks. 23

**white matter**

Brain tissue with high nerve fiber density. 5

# List of Figures

# Bibliography


Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, and M. Isard (2016). "Tensorflow: A System for Large-Scale Machine Learning". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283.

Ahrens, J., B. Geveci, and C. Law (2005). "ParaView: An End-User Tool for Large-Data Visualization". In: *Visualization Handbook*, pp. 717–731. DOI: 10.1016/B978-012387582-2/50038-1.

Amunts, K., A. Schleicher, and K. Zilles (2007a). "Cytoarchitecture of the Cerebral Cortex–More than Localization". In: *NeuroImage* 37, pp. 1061–1065. DOI: 10.1016/j.neuroimage.2007.02.037.

Amunts, K., E. Armstrong, A. Malikovic, L. Hömke, H. Mohlberg, A. Schleicher, and K. Zilles (2007b). "Gender-Specific Left-Right Asymmetries in Human Visual Cortex". In: *The Journal of Neuroscience: The Official Journal of the Society for Neuroscience* 27, pp. 1356–1364. DOI: 10.1523/JNEUROSCI.4753-06.2007.

Amunts, K., C. Lepage, L. Borgeat, H. Mohlberg, T. Dickscheid, M.-É. Rousseau, S. Bludau, P.-L. Bazin, L. B. Lewis, A.-M. Oros-Peusquens, N. J. Shah, T. Lippert, K. Zilles, and A. C. Evans (2013). "BigBrain: An Ultrahigh-Resolution 3D Human Brain Model". In: *Science* 340, pp. 1472–1475. DOI: 10.1126/science.1235381.

Amunts, K. and T. Lippert (2021). "Brain Research Challenges Supercomputing". In: *Science* 374, pp. 1054–1055. DOI: 10.1126/science.abl8519.

Amunts, K., A. Malikovic, H. Mohlberg, T. Schormann, and K. Zilles (2000). "Brodmann's Areas 17 and 18 Brought into Stereotaxic Space—Where and How Variable?" In: *NeuroImage* 11, pp. 66–84. DOI: 10.1006/nimg.1999.0516.

Amunts, K., H. Mohlberg, S. Bludau, and K. Zilles (2020). "Julich-Brain: A 3D Probabilistic Atlas of the Human Brain's Cytoarchitecture". In: *Science* 369, p. 988. DOI: 10.1126/science.abb4588.

Amunts, K., A. Schleicher, U. Bürgel, H. Mohlberg, H. B. M. Uylings, and K. Zilles (1999). "Broca's Region Revisited: Cytoarchitecture and Intersubject Variability". In: *Journal of Comparative Neurology* 412, pp. 319–341. DOI: 10.1002/(SICI)1096-9861(19990920)412:2<319::AID-CNE10>3.0.CO;2-7.

Amunts, K. and K. Zilles (2015). "Architectonic Mapping of the Human Brain beyond Brodmann". In: *Neuron* 88, pp. 1086–1107. DOI: 10.1016/j.neuron.2015.12.001.

Atzeni, A., M. Jansen, S. Ourselin, and J. E. Iglesias (2018). "A Probabilistic Model Combining Deep Learning and Multi-Atlas Segmentation for Semi-Automated Labelling of Histology". In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*, pp. 219–227. DOI: 10.1007/978-3-030-00934-2_25.

Avants, B. B., N. Tustison, and G. Song (2009). "Advanced Normalization Tools (ANTS)". In: *Insight j* 2, pp. 1–35.

Bach, S., A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek (2015). "On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation". In: *PLOS ONE* 10. DOI: 10.1371/journal.pone.0130140.

Barredo Arrieta, A., N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera (2020). "Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI". In: *Information Fusion* 58, pp. 82–115. DOI: 10.1016/j.inffus.2019.12.012.

Bay, H., T. Tuytelaars, and L. Van Gool (2006). "Surf: Speeded up Robust Features". In: *European Conference on Computer Vision*, pp. 404–417.

Blinkov, S. M. and I. G. Il'ja (1968). *Das Zentralnervensystem in Zahlen und Tabellen*.

Bludau, S. (2011). "Cytoarchitectonic Mapping of the Human Frontal Pole". PhD thesis. RWTH Aachen.

Bradler, S. H. (2015). "Multimodale Kartierung und Funktion des Sulcus frontalis inferior des menschlichen Gehirns". PhD thesis. HHU Düsseldorf.

Bradski, G. (2000). "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools*, pp. 120–123.

Brandstetter, A., N. Bolakhrif, C. Schiffer, T. Dickscheid, H. Mohlberg, and K. Amunts (2021). "Deep Learning-Supported Cytoarchitectonic Mapping of the Human Lateral Geniculate Body in the BigBrain". In: *Brain-Inspired Computing*, pp. 22–32.

Brodmann, K. (1909). *Vergleichende Lokalisationslehre der Großhirnrinde*.

Brody, S., U. Alon, and E. Yahav (2021). *How Attentive Are Graph Attention Networks?* URL: http://arxiv.org/abs/2105.14491.

Brown, T., B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei (2020). "Language Models Are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Vol. 33, pp. 1877–1901.

Bruna, J., W. Zaremba, A. Szlam, and Y. LeCun (2014). "Spectral Networks and Locally Connected Networks on Graphs". In: *International Conference on Learning Representations (ICLR 2014)*.

Carns, P., K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross (2011). "Understanding and Improving Computational Science Storage Access through Continuous Characterization". In: *ACM Transactions on Storage* 7. DOI: 10.1145/2027066.2027068.

Caron, M., I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin (2020). "Unsupervised Learning of Visual Features by Contrasting Cluster Assignments". In: *Advances in Neural Information Processing Systems*. Vol. 33, pp. 9912–9924.

Caspers, J., K. Zilles, S. B. Eickhoff, A. Schleicher, H. Mohlberg, and K. Amunts (2013). "Cytoarchitectonical Analysis and Probabilistic Mapping of Two Extrastriate Areas of the Human Posterior Fusiform Gyrus". In: *Brain Structure and Function* 218, pp. 511–526. DOI: 10.1007/s00429-012-0411-8.

Caspers, S., S. Geyer, A. Schleicher, H. Mohlberg, K. Amunts, and K. Zilles (2006). "The Human Inferior Parietal Cortex: Cytoarchitectonic Parcellation and Interindividual Variability". In: *NeuroImage* 33, pp. 430–448. DOI: 10.1016/j.neuroimage.2006.06.054.

Chan, T. and L. Vese (1999). "An Active Contour Model without Edges". In: *Scale-Space Theories in Computer Vision*, pp. 141–151. DOI: 10.1007/3-540-48236-9_13.

Chen, J., L. Yang, Y. Zhang, M. Alber, and D. Z. Chen (2016a). "Combining Fully Convolutional and Recurrent Neural Networks for 3d Biomedical Image Segmentation". In: *Advances in Neural Information Processing Systems*, pp. 3036–3044.

Chen, T., B. Xu, C. Zhang, and C. Guestrin (2016b). *Training Deep Nets with Sublinear Memory Cost*. URL: http://arxiv.org/abs/1604.06174.

Chen, T., S. Kornblith, M. Norouzi, and G. Hinton (2020). "A Simple Framework for Contrastive Learning of Visual Representations". In: *International Conference on Machine Learning (ICML 2020)*, pp. 1597–1607.

Chen, X. and K. He (2021). "Exploring Simple Siamese Representation Learning". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15750–15758.

Choi, H.-J., K. Zilles, H. Mohlberg, A. Schleicher, G. R. Fink, E. Armstrong, and K. Amunts (2006). "Cytoarchitectonic Identification and Probabilistic Mapping of Two Distinct Areas within the Anterior Ventral Bank of the Human Intraparietal Sulcus". In: *Journal of Comparative Neurology* 495, pp. 53–69. DOI: 10.1002/cne.20849.

Cignoni, P., M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia (2008). "MeshLab: An Open-Source Mesh Processing Tool". In: *Eurographics Italian Chapter Conference*.

Collette, A. (2013). *Python and HDF5*.

Collins, D. L., P. Neelin, T. M. Peters, and A. C. Evans (1994). "Automatic 3D Intersubject Registration of MR Volumetric Data in Standardized Talairach Space". In: *Journal of Computer Assisted Tomography* 18, pp. 192–205.

Creutzfeldt, O. (1995). *Cortex Cerebri: Performance, Structural and Functional Organisation of the Cortex*.

Cybenko, G. (1989). "Approximation by Superpositions of a Sigmoidal Function". In: *Mathematics of control, signals and systems* 2, pp. 303–314.

Dalcin, L. D., R. R. Paz, P. A. Kler, and A. Cosimo (2011). "Parallel Distributed Computing Using Python". In: *Advances in Water Resources* 34, pp. 1124–1139. DOI: 10.1016/j.advwatres.2011.04.013.

Defferrard, M., X. Bresson, and P. Vandergheynst (2016). "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering". In: *30th International Conference on Neural Information Processing Systems*, pp. 3844–3852.

Devakuruparan, S., A. Brandstetter, S. Bludau, M. Minnerop, H. Mohlberg, Christian Schiffer, A. Schnitzler, H. Uylings, and K. Amunts (2021). "Cytoarchitectonic Mapping and Analysis of the VIM Nucleus of the Thalamus in Ten Human Post Mortem Brains Including the BigBrain2". In: *5th BigBrain Workshop*.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (2018). *Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding*. URL: http://arxiv.org/abs/1810.04805.

Dickscheid, T., S. Bludau, C. Paquola, C. Schiffer, E. Upschulte, and K. Amunts (2021a). "Layer-Specific Distributions of Segmented Cells in Area 4a (PreCG) of BigBrain [Data Set]". In: *EBRAINS*. DOI: 10.25493/PV5Q-SFR.

— (2021b). "Layer-Specific Distributions of Segmented Cells in Area 7P (SPL) of BigBrain [Data Set]". In: *EBRAINS*. DOI: 10.25493/35K1-BTX.

— (2021c). "Layer-Specific Distributions of Segmented Cells in Area FG1 (FusG) of BigBrain [Data Set]". In: *EBRAINS*. DOI: 10.25493/E1FY-2C4.

— (2021d). "Layer-Specific Distributions of Segmented Cells in Area hOc1 (V1, 17, CalcS) of BigBrain [Data Set]". In: *EBRAINS*. DOI: 10.25493/EVPC-MVH.

— (2021e). "Layer-Specific Distributions of Segmented Cells in Area hOc4d (Cuneus) of BigBrain [Data Set]". In: *EBRAINS*. DOI: 10.25493/ZXQX-JKP.

Dickscheid, T., S. Haas, S. Bludau, P. Glock, M. Huysegoms, and K. Amunts (2019). "Towards 3D Reconstruction of Neuronal Cell Distributions from Histological Human Brain Sections". In: *Future Trends of HPC in a Disruptive Scenario* 34, p. 223.

Dijkstra, E. W. (1959). "A Note on Two Problems in Connexion with Graphs". In: *Numerische Mathematik* 1, pp. 269–271. DOI: 10.1007/BF01386390.

Ding, S.-L., J. J. Royall, S. M. Sunkin, L. Ng, B. A. C. Facer, P. Lesnar, A. Guillozet-Bongaarts, B. McMurray, A. Szafer, T. A. Dolbeare, A. Stevens, L. Tirrell, T.

Benner, S. Caldejon, R. A. Dalley, N. Dee, C. Lau, J. Nyhus, M. Reding, Z. L. Riley, D. Sandman, E. Shen, A. van der Kouwe, A. Varjabedian, M. Write, L. Zollei, C. Dang, J. A. Knowles, C. Koch, J. W. Phillips, N. Sestan, P. Wohnoutka, H. R. Zielke, J. G. Hohmann, A. R. Jones, A. Bernard, M. J. Hawrylycz, P. R. Hof, B. Fischl, and E. S. Lein (2016). "Comprehensive Cellular-Resolution Atlas of the Adult Human Brain". In: *Journal of Comparative Neurology* 524, pp. 3127–3481. DOI: 10.1002/cne.24080.

Doersch, C., A. Gupta, and A. A. Efros (2015). "Unsupervised Visual Representation Learning by Context Prediction". In: *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1422–1430.

Dosovitskiy, A., L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. (2020). "An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations (ICLR 2020)*.

Dumoulin, V. and F. Visin (2018). *A Guide to Convolution Arithmetic for Deep Learning*. URL: http://arxiv.org/abs/1603.07285.

Duyn, J. H. (2012). "The Future of Ultra-High Field MRI and fMRI for Study of the Human Brain". In: *NeuroImage* 62, pp. 1241–1248. DOI: 10.1016/j.neuroimage.2011.10.065.

Eickhoff, S. B., T. Paus, S. Caspers, M.-H. Grosbras, A. C. Evans, K. Zilles, and K. Amunts (2007). "Assignment of Functional Activations to Probabilistic Cytoarchitectonic Areas Revisited". In: *NeuroImage* 36, pp. 511–521. DOI: 10.1016/j.neuroimage.2007.03.060.

Eickhoff, S. B., A. Schleicher, K. Zilles, and K. Amunts (2006). "The Human Parietal Operculum. I. Cytoarchitectonic Mapping of Subdivisions". In: *Cerebral Cortex* 16, pp. 254–267. DOI: 10.1093/cercor/bhi105.

Eickhoff, S. B., K. E. Stephan, H. Mohlberg, C. Grefkes, G. R. Fink, K. Amunts, and K. Zilles (2005). "A New SPM Toolbox for Combining Probabilistic Cytoarchitectonic Maps and Functional Imaging Data". In: *NeuroImage* 25, pp. 1325–1335. DOI: 10.1016/j.neuroimage.2004.12.034.

Evans, A. C., A. L. Janke, D. L. Collins, and S. Baillet (2012). "Brain Templates and Atlases". In: *NeuroImage* 62, pp. 911–922. DOI: 10.1016/j.neuroimage.2012.01.024.

Fey, M. and J. E. Lenssen (2019). "Fast Graph Representation Learning with PyTorch Geometric". In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Fischl, B. (2012). "FreeSurfer". In: *NeuroImage* 62, pp. 774–781. DOI: 10.1016/j.neuroimage.2012.01.021.

Fischler, M. A. and R. C. Bolles (1981). "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Communications of the ACM* 24, pp. 381–395.

Fonov, V., A. C. Evans, K. Botteron, C. R. Almli, R. C. McKinstry, and D. L. Collins (2011). "Unbiased Average Age-Appropriate Atlases for Pediatric Studies". In: *NeuroImage* 54, pp. 313–327. DOI: 10.1016/j.neuroimage.2010.07.033.

Fukushima, K. and S. Miyake (1982). "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Visual Pattern Recognition". In: *Competition and Cooperation in Neural Nets*, pp. 267–285.

Gadermayr, M., V. Appel, B. M. Klinkhammer, P. Boor, and D. Merhof (2018). "Which Way Round? A Study on the Performance of Stain-Translation for Segmenting Arbitrarily Dyed Histological Images". In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*, pp. 165–173. DOI: 10.1007/978-3-030-00934-2_19.

Gal, Y. and Z. Ghahramani (2016). "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning". In: *International Conference on Machine Learning*, pp. 1050–1059.

Geyer, S., A. Ledberg, A. Schleicher, S. Kinomura, T. Schormann, U. Bürgel, T. Klingberg, J. Larsson, K. Zilles, and P. E. Roland (1996). "Two Different Areas within the Primary Motor Cortex of Man". In: *Nature* 382 (6594), pp. 805–807. DOI: 10.1038/382805a0.

Geyer, S., A. Schleicher, and K. Zilles (1999). "Areas 3a, 3b, and 1 of Human Primary Somatosensory Cortex: 1. Microstructural Organization and Interindividual Variability". In: *NeuroImage* 10, pp. 63–83. DOI: 10.1006/nimg.1999.0440.

Gidaris, S., P. Singh, and N. Komodakis (2018). "Unsupervised Representation Learning by Predicting Image Rotations". In: *International Conference on Learning Representations (ICLR 2018)*.

Gilmer, J., S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl (2017). "Neural Message Passing for Quantum Chemistry". In: *International Conference on Machine Learning (ICML 2017)*, pp. 1263–1272.

Glock, P. (2020). *Pytiff*. Big Data Analytics group. URL: https://github.com/FZJ-INM1-BDA/pytiff.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014). "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems*, pp. 2672–2680.

Goyal, P., P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He (2017). *Accurate, Large Minibatch SGD: Training Imagenet in 1 Hour*. URL: https://arxiv.org/abs/1706.02677.

Graf, S. and O. Mextorf (2021). "JUST: Large-Scale Multi-Tier Storage Infrastructure at the Jülich Supercomputing Centre". In: *Journal of large-scale research facilities JLSRF* 7 (0), p. 180. DOI: 10.17815/jlsrf-7-180.

Grigorescu, S., B. Trasnea, T. Cocias, and G. Macesanu (2020). "A Survey of Deep Learning Techniques for Autonomous Driving". In: *Journal of Field Robotics* 37, pp. 362–386. DOI: 10.1002/rob.21918.

Grill, J.-B., F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, B. Piot, K. Kavukcuoglu, R. Munos, and M. Valko (2020). "Bootstrap Your Own Latent: A New Approach to Self-Supervised Learning". In: *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*.

Grodzinsky, Y., I. Deschamps, P. Pieperhoff, F. Iannilli, G. Agmon, Y. Loewenstein, and K. Amunts (2020). "Logical Negation Mapped onto the Brain". In: *Brain Structure and Function* 225, pp. 19–31. DOI: 10.1007/s00429-019-01975-w.

Hadsell, R., S. Chopra, and Y. LeCun (2006). "Dimensionality Reduction by Learning an Invariant Mapping". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 2, pp. 1735–1742.

Hahnloser, R. H., R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung (2000). "Digital Selection and Analogue Amplification Coexist in a Cortex-Inspired Silicon Circuit". In: *Nature* 405, pp. 947–951.

Halchenko, Y. O., K. Meyer, B. Poldrack, D. S. Solanky, A. S. Wagner, J. Gors, D. MacFarlane, D. Pustina, V. Sochat, S. S. Ghosh, C. Mönch, C. J. Markiewicz, L. Waite, I. Shlyakhter, A. de la Vega, S. Hayashi, C. O. Häusler, J.-B. Poline, T. Kadelka, K. Skytén, D. Jarecka, D. Kennedy, T. Strauss, M. Cieslak, P. Vavra, H.-I. Ioanas, R. Schneider, M. Pflüger, J. V. Haxby, S. B. Eickhoff, and M. Hanke (2021). "DataLad: Distributed System for Joint Management of Code, Data, and Their Relationship". In: *Journal of Open Source Software* 6, p. 3262. DOI: 10.21105/joss.03262.

Hamilton, W. L. (2020). *Graph Representation Learning*.

Hamilton, W. L., R. Ying, and J. Leskovec (2017). "Inductive Representation Learning on Large Graphs". In: *31st International Conference on Neural Information Processing Systems*, pp. 1025–1035.

Harris, C. R., K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant (2020). "Array Programming with NumPy". In: *Nature* 585, pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

He, K., H. Fan, Y. Wu, S. Xie, and R. Girshick (2020). "Momentum Contrast for Unsupervised Visual Representation Learning". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9729–9738.

He, K., X. Zhang, S. Ren, and J. Sun (2016a). "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.

— (2016b). "Identity Mappings in Deep Residual Networks". In: *European Conference on Computer Vision*, pp. 630–645.

Hebb, D. O. (1949). *The Organisation of Behaviour: A Neuropsychological Theory*.

Hénaff, O. J., A. Srinivas, J. De Fauw, A. Razavi, C. Doersch, S. M. A. Eslami, and A. van den Oord (2020). "Data-Efficient Image Recognition with Contrastive Predictive Coding". In: *International Conference on Machine Learning*.

Hendricks, L. A., Z. Akata, M. Rohrbach, J. Donahue, B. Schiele, and T. Darrell (2016). "Generating Visual Explanations". In: *Computer Vision – ECCV 2016*, pp. 3–19. DOI: 10.1007/978-3-319-46493-0_1.

Henssen, A., K. Zilles, N. Palomero-Gallagher, A. Schleicher, H. Mohlberg, F. Gerboga, S. B. Eickhoff, S. Bludau, and K. Amunts (2016). "Cytoarchitecture and Probability Maps of the Human Medial Orbitofrontal Cortex". In: *Cortex* 75, pp. 87–112. DOI: 10.1016/j.cortex.2015.11.006.

Hinton, G. E. and R. R. Salakhutdinov (2006). "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313, pp. 504–507.

Hinton, G. E., N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov (2012). *Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors*. URL: https://arxiv.org/abs/1207.0580.

Holmes, C. J., R. Hoge, L. Collins, R. Woods, A. W. Toga, and A. C. Evans (–Apr. 1998). "Enhancement of MR Images Using Registration for Signal Averaging". In: *Journal of Computer Assisted Tomography* 22, pp. 324–333.

Hömke, L. (2006). "A Multigrid Method for Anisotropic PDEs in Elastic Image Registration". In: *Numerical linear algebra with applications* 13, pp. 215–229.

Huang, G., Z. Liu, L. Van Der Maaten, and K. Q. Weinberger (2017). "Densely Connected Convolutional Networks." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 1, p. 3.

Huang, Y., Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and z. Chen (2019). "GPipe: Efficient Training of Giant Neural Networks Using Pipeline Parallelism". In: *Advances in Neural Information Processing Systems*. Vol. 32.

Hunter, J. D. (2007). "Matplotlib: A 2D Graphics Environment". In: *Computing in Science Engineering* 9, pp. 90–95. DOI: 10.1109/MCSE.2007.55.

Ioffe, S. and C. Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *International Conference on Machine Learning*, pp. 448–456.

Jain, R., H. Fan, R. B. Girshick, and K. He (2020). *Improved Baselines with Momentum Contrastive Learning*. URL: https://arxiv.org/abs/2003.04297.

Jenkinson, M., C. F. Beckmann, T. E. J. Behrens, M. W. Woolrich, and S. M. Smith (2012). "FSL". In: *NeuroImage* 62, pp. 782–790. DOI: 10.1016/j.neuroimage.2011.09.015.

Jing, L. and Y. Tian (2019). *Self-Supervised Visual Feature Learning with Deep Neural Networks: A Survey*. URL: http://arxiv.org/abs/1902.06162.

Jones, S. E., B. R. Buchbinder, and I. Aharon (2000). "Three-Dimensional Mapping of Cortical Thickness Using Laplace's Equation". In: *Human Brain Mapping* 11, pp. 12–32. DOI: 10.1002/1097-0193(200009)11:1<12::aid-hbm20>3.0.co;2-k.

Joseph, K. J., S. Khan, F. S. Khan, and V. N. Balasubramanian (2021). "Towards Open World Object Detection". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5830–5840.

Jumper, J., R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis (2021). "Highly Accurate Protein Structure Prediction with AlphaFold". In: *Nature* 596 (7873), pp. 583–589. DOI: 10.1038/s41586-021-03819-2.

Kanopoulos, N., N. Vasanthavada, and R. Baker (1988). "Design of an Image Edge Detection Filter Using the Sobel Operator". In: *IEEE Journal of Solid-State Circuits* 23, pp. 358–367. DOI: 10.1109/4.996.

Kazhdan, M., M. Bolitho, and H. Hoppe (2006). "Poisson Surface Reconstruction". In: *Fourth Eurographics Symposium on Geometry Processing*, pp. 61–70.

Kendall, A. and Y. Gal (2017). "What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?" In: *31st International Conference on Neural Information Processing Systems*, pp. 5580–5590.

Khosla, P., P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan (2020). "Supervised Contrastive Learning". In: *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*.

Kiefer, J. and J. Wolfowitz (1952). "Stochastic Estimation of the Maximum of a Regression Function". In: *The Annals of Mathematical Statistics*, pp. 462–466.

Kikinis, R., S. D. Pieper, and K. G. Vosburgh (2014). "3D Slicer: A Platform for Subject-Specific Image Analysis, Visualization, and Clinical Support". In: *Intra-*

*operative Imaging and Image-Guided Therapy*, pp. 277–289. DOI: 10.1007/978-1-4614-7657-3_19.

Kingma, D. P. and J. Ba (2014). *Adam: A Method for Stochastic Optimization*. URL: https://arxiv.org/abs/1412.6980.

Kipf, T. N. and M. Welling (2017). "Semi-Supervised Classification with Graph Convolutional Networks". In: *International Conference on Learning Representations (ICLR 2017)*.

Kiwitz, K., A. Brandstetter, C. Schiffer, S. Bludau, H. Mohlberg, M. Omidyeganeh, P. Massicotte, and K. Amunts (2022). "Cytoarchitectonic Maps of the Human Metathalamus in 3D Space". In: *Frontiers in Neuroanatomy* 16. DOI: 10.3389/fnana.2022.837485.

Kiwitz, K., C. Schiffer, H. Spitzer, T. Dickscheid, and K. Amunts (2020). "Deep Learning Networks Reflect Cytoarchitectonic Features Used in Brain Mapping". In: *Scientific Reports* 10. DOI: 10.1038/s41598-020-78638-y.

Krause, D. and P. Thörnig (2018). "JURECA: Modular Supercomputer at Jülich Supercomputing Centre". In: *Journal of large-scale research facilities JLSRF* 4, A132. DOI: 10.17815/jlsrf-4-121-1.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). "Imagenet Classification with Deep Convolutional Neural Networks". In: *Advances in neural information processing systems* 25, pp. 1097–1105.

Kujovic, M., K. Zilles, A. Malikovic, A. Schleicher, H. Mohlberg, C. Rottschy, S. B. Eickhoff, and K. Amunts (2013). "Cytoarchitectonic Mapping of the Human Dorsal Extrastriate Cortex". In: *Brain Structure and Function* 218, pp. 157–172. DOI: 10.1007/s00429-012-0390-9.

Kurth, F., S. B. Eickhoff, A. Schleicher, L. Hoemke, K. Zilles, and K. Amunts (2010). "Cytoarchitecture and Probabilistic Maps of the Human Posterior Insular Cortex". In: *Cerebral Cortex* 20, pp. 1448–1461. DOI: 10.1093/cercor/bhp208.

Lam, S. K., A. Pitrou, and S. Seibert (2015). "Numba: A LLVM-based Python JIT Compiler". In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pp. 1–6. DOI: 10.1145/2833157.2833162.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86, pp. 2278–2324.

Leprince, Y., F. Poupon, T. Delzescaux, D. Hasboun, C. Poupon, and D. Rivière (2015). "Combined Laplacian-equivolumic Model for Studying Cortical Lamination with Ultra High Field MRI (7 T)". In: *2015 IEEE 18th International Symposium on Biomedical Imaging (ISBI 2015)*, pp. 580–583. DOI: 10.1109/ISBI.2015.7163940.

Lewiner, T., H. Lopes, A. W. Vieira, and G. Tavares (2003). "Efficient Implementation of Marching Cubes' Cases with Topological Guarantees". In: *Journal of Graphics Tools* 8, pp. 1–15. DOI: 10.1080/10867651.2003.10487582.

Lewis, L., C. Lepage, M. Fournier, K. Zilles, K. Amunts, and A. Evans (2014). "BigBrain: Initial Tissue Classification and Surface Extraction". In: *Annual Meeting of the Organization for Human Brain Mapping*.

Lloyd, S. (1982). "Least Squares Quantization in PCM". In: *IEEE transactions on information theory* 28, pp. 129–137.

Long, J., E. Shelhamer, and T. Darrell (2015). "Fully Convolutional Networks for Semantic Segmentation". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440.

Lorenz, S., K. S. Weiner, J. Caspers, H. Mohlberg, A. Schleicher, S. Bludau, S. B. Eickhoff, K. Grill-Spector, K. Zilles, and K. Amunts (2017). "Two New Cytoarchitectonic Areas on the Human Mid-Fusiform Gyrus". In: *Cerebral Cortex* 27, pp. 373–385. DOI: 10.1093/cercor/bhv225.

Mahalanobis, P. C. (1936). "On the Generalized Distance in Statistics". In: *Proceedings of the National Institute of Science of India*.

Malandain, G., É. Bardinet, K. Nelissen, and W. Vanduffel (2004). "Fusion of Autoradiographs with an MR Volume Using 2-D and 3-D Linear Transformations". In: *NeuroImage* 23, pp. 111–127. DOI: 10.1016/j.neuroimage.2004.04.038.

Malikovic, A., K. Amunts, A. Schleicher, H. Mohlberg, M. Kujovic, N. Palomero-Gallagher, S. B. Eickhoff, and K. Zilles (2016). "Cytoarchitecture of the Human Lateral Occipital Cortex: Mapping of Two Extrastriate Areas hOc4la and hOc4lp". In: *Brain Structure & Function* 221, pp. 1877–1897. DOI: 10.1007/s00429-015-1009-8.

Mancini, M., M. F. Naeem, Y. Xian, and Z. Akata (2021). "Open World Compositional Zero-Shot Learning". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5222–5230.

Márquez-Neila, P., L. Baumela, and L. Alvarez (2014). "A Morphological Approach to Curvature-Based Evolution of Curves and Surfaces". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, pp. 2–17. DOI: 10.1109/TPAMI.2013.106.

Marrakchi, Y., O. Makansi, and T. Brox (2021). "Fighting Class Imbalance with Contrastive Learning". In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2021*.

McCulloch, W. S. and W. Pitts (1943). "A Logical Calculus of the Ideas Immanent in Nervous Activity". In: *The bulletin of mathematical biophysics* 5, pp. 115–133.

Merker, B. (1983). "Silver Staining of Cell Bodies by Means of Physical Development". In: *Journal of Neuroscience Methods* 9, pp. 235–241. DOI: 10.1016/0165-0270(83)90086-9.

Message Passing Forum (1994). *MPI: A Message-Passing Interface Standard*. Technical Report. University of Tennessee.

Milletari, F., N. Navab, and S.-A. Ahmadi (2016). "V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation". In: *2016 Fourth International Conference on 3D Vision (3DV)*, pp. 565–571.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller (2013). *Playing Atari with Deep Reinforcement Learning*. URL: http://arxiv.org/abs/1312.5602.

Mohlberg, H., A. Schleicher, K. Zilles, S. B. Eickhoff, and K. Amunts (2012). "A new processing pipeline and release of cytoarchitectonic probabilistic maps - JuBrain". In: *18th Annual Meeting of the Organization for Human Brain Mapping*.

Mölder, F., K. P. Jablonski, B. Letcher, M. B. Hall, C. H. Tomkins-Tinch, V. Sochat, J. Forster, S. Lee, S. O. Twardziok, A. Kanitz, A. Wilm, M. Holtgrewe, S. Rahmann, S. Nahnsen, and J. Köster (2021). "Sustainable Data Analysis with Snakemake". In: *F1000Research* 10. DOI: 10.12688/f1000research.29032.1.

Morosan, P., J. Rademacher, A. Schleicher, K. Amunts, T. Schormann, and K. Zilles (2001). "Human Primary Auditory Cortex: Cytoarchitectonic Subdivisions and Mapping into a Spatial Reference System". In: *NeuroImage* 13, pp. 684–701. DOI: 10.1006/nimg.2000.0715.

Nesterov, Y. E. (1983). "A Method of Solving a Convex Programming Problem with Convergence Rate $(0(1/K^2))$". In: *Sov. Math., Dokl.* 27, pp. 372–376.

Noroozi, M. and P. Favaro (2016). "Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles". In: *European Conference on Computer Vision*, pp. 69–84.

Oden, L., C. Schiffer, H. Spitzer, T. Dickscheid, and D. Pleiter (2019). "IO Challenges for Human Brain Atlasing Using Deep Learning Methods - An In-Depth Analysis". In: *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). IEEE, pp. 291–298. DOI: 10.1109/EMPDP.2019.8671630.

Operto, G., R. Bulot, J. L. Anton, and O. Coulon (2008). "Projection of fMRI Data onto the Cortical Surface Using Anatomically-Informed Convolution Kernels". In: *NeuroImage* 39, pp. 127–135. DOI: 10.1016/j.neuroimage.2007.08.039.

Otsu, N. (1979). "A Threshold Selection Method from Gray-Level Histograms". In: *IEEE Transactions on Systems, Man, and Cybernetics* 9, pp. 62–66. DOI: 10.1109/TSMC.1979.4310076.

Palomero-Gallagher, N., S. B. Eickhoff, F. Hoffstaedter, A. Schleicher, H. Mohlberg, B. A. Vogt, K. Amunts, and K. Zilles (2015). "Functional Organization of Human Subgenual Cortical Areas: Relationship between Architectonical Segregation and

Connectional Heterogeneity". In: *NeuroImage* 115, pp. 177–190. DOI: 10.1016/j.neuroimage.2015.04.053.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*.

Patarasuk, P. and X. Yuan (2009). "Bandwidth Optimal All-Reduce Algorithms for Clusters of Workstations". In: *Journal of Parallel and Distributed Computing* 69, pp. 117–124.

Pearson, K. (1901). "On Lines and Planes of Closest Fit to Systems of Points in Space". In: *The London, Edinburgh, and Dublin philosophical magazine and journal of science* 2, pp. 559–572.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay (2011). "Scikit-Learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Petit, O., N. Thome, A. Charnoz, A. Hostettler, and L. Soler (2018). "Handling Missing Annotations for Semantic Segmentation with Deep Convnets". In: *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, pp. 20–28.

Pizer, S. M., E. P. Amburn, J. D. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. ter Haar Romeny, J. B. Zimmerman, and K. Zuiderveld (1987). "Adaptive Histogram Equalization and Its Variations". In: *Computer vision, graphics, and image processing* 39, pp. 355–368.

Pohlen, T., A. Hermans, M. Mathias, and B. Leibe (2017). "Full-Resolution Residual Networks for Semantic Segmentation in Street Scenes". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'17)*.

Polyak, B. (1964). "Some Methods of Speeding up the Convergence of Iteration Methods". In: *Ussr Computational Mathematics and Mathematical Physics* 4, pp. 1–17. DOI: 10.1016/0041-5553(64)90137-5.

Pumma, S., M. Si, W.-c. Feng, and P. Balaji (2017). "Towards Scalable Deep Learning via I/O Analysis and Optimization". In: *2017 IEEE 19th International Conference on High Performance Computing and Communications*, pp. 223–230. DOI: 10.1109/HPCC-SmartCity-DSS.2017.29.

Rademacher, J., P. Morosan, T. Schormann, A. Schleicher, C. Werner, H. J. Freund, and K. Zilles (2001). "Probabilistic Mapping and Volume Measurement of Human Primary Auditory Cortex". In: *NeuroImage* 13, pp. 669–683. DOI: 10.1006/nimg.2000.0714.

Ramesh, A., M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever (2021). "Zero-Shot Text-to-Image Generation". In: *38th International Conference on Machine Learning*.

Redmon, J., S. Divvala, R. Girshick, and A. Farhadi (2016). "You Only Look Once: Unified, Real-Time Object Detection". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788.

Al-Rfou, R., G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, and A. Belopolsky (2016). *Theano: A Python Framework for Fast Computation of Mathematical Expressions*. URL: https://arxiv.org/abs/1605.02688.

Richter, M., K. Amunts, H. Mohlberg, S. Bludau, S. B. Eickhoff, K. Zilles, and S. Caspers (2019). "Cytoarchitectonic Segregation of Human Posterior Intraparietal and Adjacent Parieto-Occipital Sulcus and Its Relation to Visuomotor and Cognitive Functions". In: *Cerebral Cortex* 29, pp. 1305–1327. DOI: 10.1093/cercor/bhy245.

Rivière, D., D. Geffroy, I. Denghien, N. Souedet, and Y. Cointepas (2009). "BrainVISA: An Extensible Software Environment for Sharing Multimodal Neuroimaging Data and Processing Tools". In: *NeuroImage* 47, S163. DOI: 10.1016/S1053-8119(09)71720-3.

Robbins, H. and S. Monro (1951). "A Stochastic Approximation Method". In: *The annals of mathematical statistics*, pp. 400–407.

Ronneberger, O., P. Fischer, and T. Brox (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28.

Roscher, R., B. Bohn, M. F. Duarte, and J. Garcke (2020). "Explainable Machine Learning for Scientific Insights and Discoveries". In: *IEEE Access* 8, pp. 42200–42216. DOI: 10.1109/ACCESS.2020.2976199.

Rosenblatt, F. (1958). "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain." In: *Psychological review* 65, p. 386.

Rottschy, C., S. B. Eickhoff, A. Schleicher, H. Mohlberg, M. Kujovic, K. Zilles, and K. Amunts (2007). "Ventral Visual Cortex in Humans: Cytoarchitectonic Mapping of Two Extrastriate Areas". In: *Human Brain Mapping* 28, pp. 1045–1059. DOI: 10.1002/hbm.20348.

Ruan, J., S. Bludau, N. Palomero-Gallagher, S. Caspers, H. Mohlberg, S. B. Eickhoff, R. J. Seitz, and K. Amunts (2018). "Cytoarchitecture, Probability Maps, and Functions of the Human Supplementary and Pre-Supplementary Motor Areas". In: *Brain Structure and Function* 223, pp. 4169–4186. DOI: 10.1007/s00429-018-1738-6.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). "Learning Representations by Back-Propagating Errors". In: *Nature* 323, pp. 533–536. DOI: 10.1038/323533a0.

Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei (2015). "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision* 115, pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

Scarselli, F., M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini (2008). "The Graph Neural Network Model". In: *IEEE transactions on neural networks* 20, pp. 61–80.

Scheperjans, F., K. Hermann, S. B. Eickhoff, K. Amunts, A. Schleicher, and K. Zilles (2008). "Observer-Independent Cytoarchitectonic Mapping of the Human Superior Parietal Cortex". In: *Cerebral Cortex* 18, pp. 846–867. DOI: 10.1093/cercor/bhm116.

Schiffer, C., K. Amunts, S. Harmeling, and T. Dickscheid (2021a). "Contrastive Representation Learning For Whole Brain Cytoarchitectonic Mapping In Histological Human Brain Sections". In: *2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI 2021)*, pp. 603–606. DOI: 10.1109/ISBI48211.2021.9433986.

Schiffer, C., A. Brandstetter, N. Bolakhrif, H. Mohlberg, and K. Amunts (2021b). "Ultrahigh Resolution 3D Cytoarchitectonic Map of the LGB (Lam 1-6, CGL, Metathalamus) Created by a Deep-Learning Assisted Workflow [Data Set]". In: *EBRAINS*. DOI: 10.25493/33Z0-BX.

Schiffer, C., S. Harmeling, K. Amunts, and T. Dickscheid (2021c). "2D Histology Meets 3D Topology: Cytoarchitectonic Brain Mapping with Graph Neural Networks". In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2021*, pp. 395–404. DOI: 10.1007/978-3-030-87237-3_38.

Schiffer, C., K. Kiwitz, K. Amunts, and T. Dickscheid (2019a). "Ultrahigh Resolution 3D Cytoarchitectonic Map of Area hOc1 (V1, 17, CalcS) Created by a Deep-Learning Assisted Workflow [Data Set]". In: *EBRAINS*. DOI: 10.25493/DGEZ-Q93.

— (2019b). "Ultrahigh Resolution 3D Cytoarchitectonic Map of Area hOc2 (V2, 18) Created by a Deep-Learning Assisted Workflow [Data Set]". In: *EBRAINS*. DOI: 10.25493/FVBY-84C.

— (2020a). "Ultrahigh Resolution 3D Cytoarchitectonic Map of Area hOc3v (LingG) Created by a Deep-Learning Assisted Workflow [Data Set]". In: *EBRAINS*. DOI: 10.25493/PH0C-EKX.

— (2020b). "Ultrahigh Resolution 3D Cytoarchitectonic Map of Area hOc5 (LOC) Created by a Deep-Learning Assisted Workflow [Data Set]". In: *EBRAINS*. DOI: 10.25493/2V62-TTG.

Schiffer, C., K. Kiwitz, A. Brandstetter, H. Mohlberg, K. Amunts, and T. Dickscheid (2021d). "Ultrahigh Resolution 3D Cytoarchitectonic Map of the Human Corpus Geniculatum Mediale (CGM, MGB, MGBv, MGBd, MGBm, Metathalamus) [Data Set]". In: *EBRAINS*. DOI: 10.25493/PNY0-NCW.

Schiffer, C., L. Schuhmacher, K. Amunts, and T. Dickscheid (2021e). "Learning to Predict Cutting Angles from Histological Human Brain Sections". In: *Medical Imaging with Deep Learning (MIDL 2021)*.

Schiffer, C., H. Spitzer, K. Kiwitz, N. Unger, K. Wagstyl, A. C. Evans, S. Harmeling, K. Amunts, and T. Dickscheid (2021f). "Convolutional Neural Networks for Cytoarchitectonic Brain Mapping at Large Scale". In: *NeuroImage* 240. DOI: 10.1016/j.neuroimage.2021.118327.

Schleicher, A., K. Amunts, S. Geyer, P. Morosan, and K. Zilles (1999). "Observer-Independent Method for Microstructural Parcellation of Cerebral Cortex: A Quantitative Approach to Cytoarchitectonics". In: *NeuroImage* 9, pp. 165–177. DOI: 10.1006/nimg.1998.0385.

Schleicher, A. and K. Zilles (2005). "The Verticality Index: A Quantitative Approach to the Analysis of the Columnar Arrangement of Neurons in the Primate Neocortex". In: *Neocortical modularity and the cell minicolumn*, pp. 181–185.

Schober, M., M. Axer, M. Huysegoms, N. Schubert, K. Amunts, and T. Dickscheid (2016). "Morphing Image Masks for Stacked Histological Sections Using Laplace's Equation". In: *Bildverarbeitung für die Medizin 2016*, pp. 146–151. DOI: 10.1007/978-3-662-49465-3_27.

Selvaraju, R. R., M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra (2017). "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization." In: *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 618–626.

Senior, A. W., R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Žídek, A. W. R. Nelson, A. Bridgland, H. Penedones, S. Petersen, K. Simonyan, S. Crossan, P. Kohli, D. T. Jones, D. Silver, K. Kavukcuoglu, and D. Hassabis (2020). "Improved Protein Structure Prediction Using Potentials from Deep Learning". In: *Nature* 577 (7792), pp. 706–710. DOI: 10.1038/s41586-019-1923-7.

Sigl, B. (2018). "Zytoarchitektur, Netzwerke und Funktionen der Areale des menschlichen dorsolateralen prämotorischen Kortex - Komponenten motorischer Planung und Kandidat für das Frontale Augenfeld". PhD thesis. HHU Düsseldorf.

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis (2016). "Mastering the Game of Go

with Deep Neural Networks and Tree Search". In: *Nature* 529 (7587), pp. 484–489. DOI: 10.1038/nature16961.

Spitzer, H. (2020). "Automatic Analysis of Cortical Areas in Whole Brain Histological Sections Using Convolutional Neural Networks". PhD thesis. HHU Düsseldorf.

Spitzer, H., K. Amunts, S. Harmeling, and T. Dickscheid (2017). "Parcellation of Visual Cortex on High-Resolution Histological Brain Sections Using Convolutional Neural Networks". In: *2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017)*, pp. 920–923. DOI: 10.1109/ISBI.2017.7950666.

— (2018a). "Compact Feature Representations for Human Brain Cytoarchitecture Using Self-Supervised Learning". In: *Medical Imaging with Deep Learning (MIDL 2018)*.

Spitzer, H., K. Kiwitz, K. Amunts, S. Harmeling, and T. Dickscheid (2018b). "Improving Cytoarchitectonic Segmentation of Human Brain Areas with Self-supervised Siamese Networks". In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*, pp. 663–671.

Springenberg, J., A. Dosovitskiy, T. Brox, and M. Riedmiller (2015). *Striving for Simplicity: The All Convolutional Net*. URL: https://arxiv.org/abs/1412.6806.

Stephan, H., W. von Möllendorff, and W. Bargmann (1975). *Handbuch der mikroskopischen Anatomie des Menschen*.

Surazhsky, V. and C. Gotsman (2003). "Explicit Surface Remeshing". In: *Eurographics Symposium on Geometry Processing*.

Sutskever, I., J. Martens, G. Dahl, and G. Hinton (2013). "On the Importance of Initialization and Momentum in Deep Learning". In: *International Conference on Machine Learning*, pp. 1139–1147.

The HDF Group (1997). *Hierarchical Data Format, Version 5*. URL: https://www.hdfgroup.org.

Trepel, M. (2017). *Neuroanatomie: Struktur und Funktion*.

Tzeng, E., J. Hoffman, N. Zhang, K. Saenko, and T. Darrell (2014). *Deep Domain Confusion: Maximizing for Domain Invariance*. URL: http://arxiv.org/abs/1412.3474.

Upschulte, E., S. Harmeling, K. Amunts, and T. Dickscheid (2022). "Contour Proposal Networks for Biomedical Instance Segmentation". In: *Medical Image Analysis*. DOI: 10.1016/j.media.2022.102371.

Van den Oord, A., Y. Li, and O. Vinyals (2019). *Representation Learning with Contrastive Predictive Coding*. URL: http://arxiv.org/abs/1807.03748.

Van der Maaten, L. and G. Hinton (2008). "Visualizing Data Using T-SNE." In: *Journal of machine learning research* 9.

Van der Walt, S., J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and t. scikit-image contributors (2014). "Scikit-Image: Image Processing in Python". In: *PeerJ* 2, e453. DOI: 10.7717/peerj.453.

Van Essen, D. C., S. M. Smith, D. M. Barch, T. E. J. Behrens, E. Yacoub, and K. Ugurbil (2013). "The WU-Minn Human Connectome Project: An Overview". In: *NeuroImage* 80, pp. 62–79. DOI: 10.1016/j.neuroimage.2013.05.041.

Van Rossum, G. (1995). *Python Tutorial*. Centrum voor Wiskunde en Informatica (CWI).

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin (2017). "Attention Is All You Need". In: *Advances in Neural Information Processing Systems 30*, pp. 5998–6008.

Veličković, P., G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio (2018). *Graph Attention Networks*. URL: http://arxiv.org/abs/1710.10903.

Vinyals, O., I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver (2019). "Grandmaster Level in StarCraft II Using Multi-Agent Reinforcement Learning". In: *Nature* 575 (7782), pp. 350–354. DOI: 10.1038/s41586-019-1724-z.

Virtanen, P., R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors (2020). "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17, pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

Vogt, C. and O. Vogt (1919). *Allgemeine Ergebnisse unserer Hirnforschung*. Vol. 21.

Von Economo, C. (1925). *Die Cytoarchitektonik der Hirnrinde des erwachsenen Menschen*.

Wagstyl, K., S. Larocque, G. Cucurull, C. Lepage, J. P. Cohen, S. Bludau, N. Palomero-Gallagher, L. B. Lewis, T. Funck, and H. Spitzer (2020). "BigBrain 3D Atlas of Cortical Layers: Cortical and Laminar Thickness Gradients Diverge in Sensory and Motor Cortices". In: *PLOS Biology* 18, e3000678.

Ward Jr, J. H. (1963). "Hierarchical Grouping to Optimize an Objective Function". In: *Journal of the American statistical association* 58, pp. 236–244.

Waskom, M. L. (2021). "Seaborn: Statistical Data Visualization". In: *Journal of Open Source Software* 6, p. 3021. DOI: 10.21105/joss.03021.

Weinberger, K. Q. and L. K. Saul (2009). "Distance Metric Learning for Large Margin Nearest Neighbor Classification." In: *Journal of machine learning research* 10.

Wes McKinney (2010). "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

Wojtasik, M. (2020). "Zytoarchitektonische Charakterisierung und funktionelle Dekodierung des lateralen orbitofrontalen Kortex im humanen Gehirn". PhD thesis. HHU Düsseldorf.

Wu, Z., Y. Xiong, S. X. Yu, and D. Lin (2018). "Unsupervised Feature Learning via Non-parametric Instance Discrimination". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3733–3742. DOI: 10.1109/CVPR.2018.00393.

Wu, Z., S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu (2020). "A Comprehensive Survey on Graph Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21. DOI: 10.1109/TNNLS.2020.2978386.

You, J., Z. Ying, and J. Leskovec (2020). "Design Space for Graph Neural Networks". In: *Advances in Neural Information Processing Systems* 33.

You, Y., I. Gitman, and B. Ginsburg (2017). *Large Batch Training of Convolutional Networks*. URL: http://arxiv.org/abs/1708.03888.

Zachary, W. W. (1977). "An Information Flow Model for Conflict and Fission in Small Groups". In: *Journal of Anthropological Research* 33, pp. 452–473. DOI: 10.1086/jar.33.4.3629752.

Zachlod, D., B. Rüttgers, S. Bludau, H. Mohlberg, R. Langner, K. Zilles, and K. Amunts (2020). "Four New Cytoarchitectonic Areas Surrounding the Primary and Early Auditory Cortex in Human Brains". In: *Cortex* 128, pp. 1–21. DOI: 10.1016/j.cortex.2020.02.021.

Zeiler, M. D. and R. Fergus (2014). "Visualizing and Understanding Convolutional Networks". In: *European Conference on Computer Vision*. DOI: 10.1007/978-3-319-10590-1_53.

Zilles, K. and K. Amunts (2012). "Architecture of the Human Cerebral Cortex". In: *The Human Nervous System*, pp. 826–885. DOI: 10.1016/B978-012547626-3/50028-4.

Zilles, K., N. Palomero-Gallagher, and K. Amunts (2015). "Cytoarchitecture and Maps of the Human Cerebral Cortex". In: *Brain Mapping*, pp. 115–135. DOI: 10.1016/B978-0-12-397025-1.00207-4.

Zilles, K. and K. Amunts (2010). "Centenary of Brodmann's Map–Conception and Fate". In: *Nature Reviews Neuroscience* 11, pp. 139–145. DOI: 10.1038/nrn2776.

Zilles, K. and G. Rehkämper (2013). *Funktionelle Neuroanatomie: Lehrbuch und Atlas*.

Zitnik, M. and J. Leskovec (2017). "Predicting Multicellular Function through Multi-Layer Tissue Networks". In: *Bioinformatics* 33, pp. 190–198. DOI: 10.1093/bioinformatics/btx252.

# Eidesstattliche Erklärung

Ich versichere an Eides Statt, dass die Dissertation von mir selbständig und ohne unzulässige fremde Hilfe unter Beachtung der "Grundsätze zur Sicherung guter Wissenschaftlicher Praxis an der Heinrich-Heine-Universität Düsseldorf" erstellt worden ist.

Ort, Datum

Unterschrift