FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Section Detection in an Automatic Pipeline for Mapping Brain Activation in the Mouse

Pedro Gonçalves Neto



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Luís Teixeira

Co-supervisor: Maryam Sadeghi

Co-supervisor: Georg Göbel

July 23, 2021

Section Detection in an Automatic Pipeline for Mapping Brain Activation in the Mouse

Pedro Gonçalves Neto

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Prof. Jorge Silva
External Examiner: Prof. António Cunha
Supervisor: Prof. Luís Teixeira

July 23, 2021

Abstract

Understanding the connection between brain activation and behaviors is a focal objective of systems neuroscience. Within these research projects, mapping brain activation consists of analyzing a brain by slicing it into very thin histological sections and counting activated neurons in each brain region. Matching experimentally-obtained histological images to their corresponding atlas is a necessary step to register the results. The recent improvements in Computer Vision tasks through the use of deep learning methods show promising results in slice-to-volume mapping. This project aims to pair a 2D coronal slice to its corresponding section in the Allen Mouse Brain Atlas by implementing a Convolutional Neural Network (CNN) pipeline. Through a reference brain model, a set of sequential coronal sections can be extracted and ordered by section number. The proposed CNN pipeline is capable of classifying the section number of an experimentally-obtained image of a brain slice and generating the corresponding atlas. The CNN pipeline is robust against distortions common in experimental coronal sections and it predicts the sectioning angle. The developed tool can be integrated into an automatic pipeline for mapping brain activation, with the capacity to replace manual labor. When performed manually, section detection takes several hours per brain and our tool reduces it to 1.8 seconds per brain section. Our method successfully performs section detection with an error of less than 1%, allows for atlas generation based on a 3D model with coronal tilts and custom regions, and, it predicts the section's cutting plane.

CCS concepts:
Computing methodologies→Artificial intelligence→Computer vision→
Computer vision problems→Matching;
Computing methodologies→Machine learning→Machine learning approaches →Neural networks;
Applied computing→Life and medical sciences→Bioinformatics

Keywords: Deep learning; Slice-to-volume; Coronal slice; Allen Mouse Brain Atlas; Histological image; Image registration

Acknowledgements

I would like to thank Maryam Sadeghi for giving me the opportunity to join this project and for all the guidance and help that made this work possible. To Professor Luís Teixeira and Professor Georg Göbel for the collaboration that allowed me to pursue my research.

I would also like to express my gratitude to my parents, who motivated me and supported me in all the decisions that led me here and to my friends, who made these years of studying much more than that.

Pedro Neto

"Our responsibility is to do what we can,
learn what we can, improve the solutions, and pass them on."

Richard P. Feynman

Contents

1 Introduction **1**
- 1.1 Systems Neuroscience Objective . . . 1
- 1.2 Motivation for an Automatic Pipeline . . . 2
- 1.3 Section Detection . . . 2
- 1.4 Research Questions . . . 3
- 1.5 Document Structure . . . 4

2 Background Knowledge **5**
- 2.1 Mapping Brain Activation in the Mouse . . . 5
- 2.2 Deep Learning Background . . . 6
  - 2.2.1 Deep Learning and Artificial Neural Networks . . . 6
  - 2.2.2 Convolutional Neural Network . . . 7
  - 2.2.3 Image Segmentation . . . 8
  - 2.2.4 Training Procedure Refinements for CNNs . . . 9

3 State of the Art **13**
- 3.1 Automatic Medical Image Processing Pipelines . . . 13
- 3.2 Slice-to-Volume Techniques . . . 14
  - 3.2.1 Matching Criterion . . . 14
  - 3.2.2 Sectioning Angle Inference . . . 15
  - 3.2.3 Similar CNN Work . . . 16
- 3.3 Summary . . . 18

4 Section Detection Specification **19**
- 4.1 Problem Specification . . . 19
- 4.2 Available Data . . . 22
  - 4.2.1 Quadrant Labelling System . . . 23
  - 4.2.2 Data Deformations . . . 25
- 4.3 Proposed Pipeline . . . 27

5 CNN-based Section Detection **33**
- 5.1 Single Label Classifier and Quadrant Label Classifier . . . 33
  - 5.1.1 Single Label Classifier Overview . . . 33
  - 5.1.2 Quadrant Label Classifier Overview . . . 34
  - 5.1.3 Data Pre-Processing and Augmentation . . . 36
  - 5.1.4 Model Architecture . . . 38
- 5.2 Group C Segmentation . . . 39

6 Atlas Generation **43**
6.1 Allen Mouse Brain Atlas 43
6.2 Color Code Generation and Depth Specification 45
6.3 Quadrant Labeling to Atlas Values 47
6.4 Plane Extraction 49
6.4.1 Single Plane Extraction 49
6.4.2 Multi-Plane Extraction 51

7 Results and Discussion **53**
7.1 Single Label Classifier Results 53
7.2 Quadrant Label Classifier Results 56
7.3 Group C Segmentation Results 58
7.4 Pipeline Evaluation 58

8 Conclusion **61**
8.1 Conclusion 61
8.2 Future Work 63
8.3 Main Contributions 64

References **65**

List of Figures

1.1 Matching experimentally-obtained brain slices (top) with the corresponding brain reference atlas (bottom). Image credit: Allen Institute [18] 3

2.1 Artificial neural network architecture with input, output and hidden layer [11] . . 7
2.2 U-Net architecture, representing downscalling, upscalling and concatenating steps. Image credit: Ronneberger *et al.* [40] 9

3.1 Representation of sectioning angle variations in a saggital brain atlas, a) perpendicular angle resulting in a perfect coronal slice b) angle with an error resulting in a superior-inferior tilt in the slice. Image credit: Allen Institute [18] 16

4.1 Example of section number 98 containing only brain stem 21
4.2 Example of section number 97 containing the brain stem and right cerebrum . . . 22
4.3 Example of section number 95 without missing regions 23
4.4 Quadrant splitting and atlas generation in different sections. Most sections are split into four quadrants (top) while sections between 84 and 102 are split into five quadrants (bottom) to generate the appropriate atlas 24
4.5 Data distribution throughout the classes in Dataset 1 25
4.6 Data distribution throughout the classes in Dataset 2 26
4.7 Data distribution throughout the classes in Dataset 3 27
4.8 Quadrant distribution in Group A sections 28
4.9 Different values in quadrant distribution in Group C sections, showing an off-plane brain stem with a Q5 of 83, while the cerebrum regions suggest an α-rotation . . 29
4.10 Contrast and color variations in brain slice images. Sections can have poor light conditions, making it challenging to analyze them without applying contrast corrections (top) 29
4.11 a) Section number 66 example containing significant tears, missing tissue and luminous spots b) Section number 66 example containing little artifacts c) Section number 26 example with missing region on the top-right quadrant (Q3) d) Section number 85 example with faded regions in the right quadrants 30
4.12 Regions of a Group C section 30
4.13 Representation of the proposed pipeline 31

5.1 Effects of different contrast normalization functions 37
5.2 Group C section and corresponding ground truth mask 40

5.3 Segmenting process from predicting the mask, cleaning outliers and box generation. (1) Original image; (2) Predicted noisy mask that fits the general shape but with small incorrect patches; (3) Processed mask with at most one region from each category; (4) Boundary boxes for each region; (5) Re-scaled mask with region boundaries that fit the original image; (6) Region boundaries applied to the original section 41

6.1 Object structure for Main Olfactory Bulb (MOB) region obtained with the Allen's API [18] 44
6.2 Somatosensory areas (SS) in highlight, without distinction between child regions. Image credit: Allen Institute of Brain Science [18] 45
6.3 Effects of depth estimation in region detail 45
6.4 Region structure with custom color code considering specified visible regions . . 46
6.5 Merging relation map with visible regions color map, resulting in a color map with all the regions 47
6.6 Quadrants and angles in a section atlas 48
6.7 Converting quadrant values to angles. Q1, Q2, Q3 and, Q4 correspond to the quadrant values and SL to the section number predicted in the Single Label Classifier 49
6.8 Representation of z-coordinate variations in row belonging to a atlas with an α-angle tilt 50
6.9 Single plane atlas with left-right differentiation 51
6.10 Multi plane atlas with left-right differentiation 52

7.1 Learning rate finder of a MobileNetV2 model applied to single label regression . 55
7.2 Learning rate scheduler for single label regression 56
7.3 Results of atlas generation. The final tool takes experimentally-obtained section images (left) and automatically generates the appropriate atlas (right) 60

List of Tables

3.1 Overview of state of the art 18

7.1 Pre-trained model's performance on a test dataset when training all the layers against train only the top layers 54
7.2 Performance of different architectures on a test dataset 54
7.3 Results of single label classifier 55
7.4 Comparison of performance between two differently sized architectures 57
7.5 Comparison of performance between two different learning rate setup in the same model, where LR1 corresponds to a cosine decay starting with a default value of 0.025 and LR2 corresponds to one cycle policy with minimum and maximum values of $2 \times 1 \times 10^{-6}$ and $2 \times 1 \times 10^{-5}$ 57
7.6 Final results of quadrant label regression 57
7.7 Results of quadrant label regression without random eraser 58

Abbreviations

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
FP16	16-bit floating point
FP32	32-bit floating point
MAE	Mean Absolute Error
MB	Megabytes
NRRD	Nearly Raw Raster Data
ReLU	Rectified Linear Units
RGB	Red-Green-Blue
SNN	Siamese Neural Network
STPT	Serial Two-Photon Tomography

Chapter 1

Introduction

Understanding the connection between mind and brain has proven to be a challenging problem. Systems neuroscience aims to solve this question by discovering the relations between brain activation and behaviors. To do so, it focuses on linking the responsible neural circuit to its sensory experience. Throughout this thesis, we directed our efforts into contributing for a tool that facilitates the development of systems neuroscience projects. We deal with section detection in coronal slices, matching experimental mouse brain images to their corresponding atlas. Our work will be done together with the Medizinische Universität Innsbruck [1], where software for mapping brain activation in the mouse is currently being developed. We are contributing to this software, creating the automatic section number detection step.

1.1 Systems Neuroscience Objective

The work developed integrates an automatic pipeline using mouse brain images at a cellular resolution to map brain activation originated by social behavior. This automatic method will enlighten the connection between emotional experiences and the neural circuits that cause them. In order to visualize brain activation, it is necessary to use brain activation markers. With the immediate-early-gene c-fos marker, which was the marker selected in this method, it is possible to observe the c-fos-GFP+ neurons by fluorescent imaging. The images containing the fluorescent neurons are used in the pipeline since the fluorescence makes computational detection of the interest neurons possible.

In order to achieve the final goal, it is necessary to understand where the brain activation occurred. The pipeline receives a sequence of images of coronal slices at a single-cell resolution. The sequence is obtained by slicing a mouse brain, meaning that the set of images presents the full brain through a two-dimensional view. Each coronal slice contains different regions of the brain; thus, it is necessary to identify and match each of these regions in the image. By doing so, it allows performing the image registration step, which maps the slice to the corresponding atlas and obtaining a single coordinate system containing brain region information within that slice. It

is then possible to collect the necessary information about active neurons in each region. Once this information is obtained, the neuron count is statistically analyzed, and conclusions can be drawn.

This thesis focuses on developing the step prior to image registration, which is the task of attributing to each image an atlas representative of the brain areas existent in that individual slice. This is accomplished by pairing an experimentally-obtained slice with an atlas slice obtained from a reference brain containing the necessary information about region distribution throughout the brain. We call this step section detection.

1.2 Motivation for an Automatic Pipeline

The interest in developing the tool revolves around the drastic reduction of human labor that it brings. Currently, the usual procedure of similar projects is to manually pair the experimental images with the corresponding atlas and count the activated neurons in each region. The human workload is extensive, and the duration of such projects is long. The proposed tool aims at obtaining the same results in a fraction of the time and effort previously needed.

The recent improvements in computer vision show promising results in medical imaging problems similar to this. Particularly, the evolution of machine learning techniques such as Convolution Neural Networks (CNN) has resulted in state-of-the-art solutions both in classification and regression challenges.

1.3 Section Detection

The focus of this thesis is to develop the experimental section to atlas matching step, which must solve the slice-to-volume integration. We will refer to this step as section detection. To accomplish it, we explore CNN-based solutions. The proposed model must be robust and capable of dealing with data issues. Since the images are experimentally-obtained, they often contain unwanted artifacts such as missing tissue, folding, air bubbles, and holes. Data augmentation methods are also used in order to both enlarge the dataset and strengthen the model. Slicing the coronal sections may result in an undesired angle, which the model must also be able to deal with. Figure 1.1 illustrates the matching operation we focus on, with the real images on the top row and atlases on the bottom.

The problem can be seen either as a classification or regression problem. Since the slices are numbered, we can see the matching as a classification problem with the goal to attribute the correct label - slice number - to each section. Another view is to see the atlas numbers as a continuous label and thus, a regression problem.

Section detection is a very time-consuming task. Performing it in a single brain takes many hours and it can only be accurately performed by a clinical expert. With our automatic section detection tool, classifying a full brain takes between one and four minutes - depending on the number of slices taken from that brain - and it does not require any intervention from an experienced neuro-scientist.

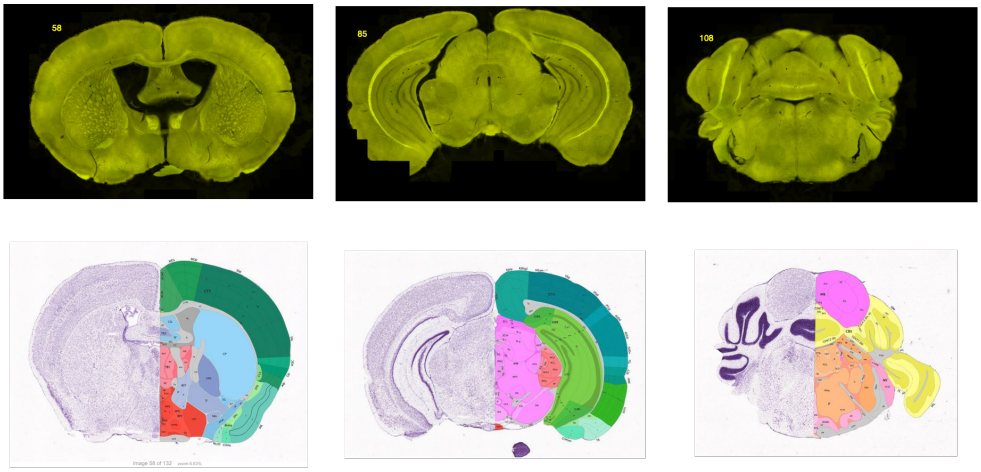


Figure 1.1: Matching experimentally-obtained brain slices (top) with the corresponding brain reference atlas (bottom). Image credit: Allen Institute [18]

Extracting the atlases is an important complement to section detection. The atlas generation algorithm not only makes the experimental slice to atlas matching fully automatic but also provides a powerful tool to the clinical experts. With it, it is possible to fully customize the atlases in both plane angles and region detail, which is a significant improvement from the restrictive use of pre-generated atlases that was previously the only option.

In the end, we expect to obtain a model capable of pairing the correct reference atlas to each coronal slice at a human accuracy level.

1.4 Research Questions

In order to guide our work and to set research goals we aim to accomplish, the following research questions (RQ) were formulated:

- **RQ1: Can a deep learning approach replace manual section detection?**
  Manually performing section detection is a laborious and slow task. Developing a deep learning-based software that achieves similar results significantly decreases human effort;
- **RQ2: Can custom mouse brain atlases be generated?**
  Using the available 2D atlases has limitations, such as a fixed cutting angle and region detail. Can another method be used to customize the atlases to fit particular projects?
- **RQ3: Can the cutting angle be inferred from a brain section using deep learning?**
  Errors while cutting the brain slices can result in tilted sections. Can this angle be captured from a brain section image and what methods can accomplish this task?

- **RQ4: How does deep learning-based section detection perform with deformed brain sections?**
  Since brain sections often contain large deformations, robustness is a substantial aspect of section detection solutions;
- **RQ5: What are the relevant factors to improve and develop deep learning models?**
  Since deep learning models have multiple variables, it is relevant to identify an efficient method and the most important variables to improve the models.

1.5 Document Structure

We will start by presenting a literature review on the topic, which provides inspiration for the methods used as well as more extensive knowledge of the domain. In Chapter 2 we present the background knowledge relevant to understanding our work and, in Chapter 3, we discuss the state of the art in section detection.

A concrete problem specification will then be presented in Chapter 4. In this chapter, we will formulate section detection as a deep learning task and introduce our proposed pipeline. The chapters following our problem formulation, Chapter 5 and 6, describe the methods used for each step of the pipeline.

Finally, in Chapter 7 the results are presented along with the process that leads to them. The solution is discussed and the conclusions are drawn in Chapter 8.

Chapter 2

Background Knowledge

In this chapter, we will begin by contextualizing the focus of our work by further illustrating the pipeline's workflow and context. Our task's success heavily depends on the proper use of deep learning techniques - with a particular focus on CNN - and thus, we present a background of deep learning development pertinent to our field of work.

2.1 Mapping Brain Activation in the Mouse

As previously mentioned in Chapter 1, there is a relevant interest in developing an automatic tool for mapping brain activation. Systems neuroscience knowledge can be further deepened by understanding the relationship between brain activation and sensory experience. Since such tasks are labor-intensive, the recent developments in computer vision propel the automation of these efforts. The Department of Medical Statistics, Informatics and Health Economics at the Medizinische Universität Innsbruck [1] is currently working on an automatic pipeline for mouse brain mapping in the mouse through the use of histological images.

To properly contextualize this work, we will elaborate on the clinical objective the automatic system aims to assist with. The connection between neural circuits and emotional-valence behaviors has been studied in the previous years. However, the linkage between brain activation and salience or novelty-driven behaviors is still a topic to be further understood. Salient events can contain both positive and negative information. Detecting them and reacting to them is important for an individuals' survival.

Regarding the computational pipeline, a series of methods allowed for the detection of the c-fos-GFP+. After that, their distribution in brain regions is registered, and the neuron counts are statistically analyzed. In order to achieve it, the pipeline is divided into four phases: cell detection and count, brain section to reference atlas registration, brain area cell localization, and section number detection.

Counting activated (fluorescent) neurons is a challenging task due to inconsistencies in size, intensity, and shape. In the developed pipeline, neurons are automatically detected for each color

channel separately, and the total detected neurons are stored along with their coordinates, which will be necessary for the next step.

Automatically registering the reference atlas to the histological images would be the ideal scenario since it would generate the perfect setup to compare and integrate the results of each anatomical brain. Unfortunately, mapping the histological images to their reference atlas is a defying task in terms of accuracy, versatility, and time efficiency. Since the images are experimentally-obtained, they often contain deformations that impede the use of strict registration methods.

After obtaining a correct distribution of the brain areas present in each slice, each neuron's coordinates - discovered in the first step - can be used to find the neuron's location in the brain. The neurons are then projected in the registered atlas, and the amount of activated neurons in each brain region is counted.

This thesis focuses on section detection, solving an issue preceding the step of registering the image to the atlas. In order to correctly register the brain regions to the histological slices, it is necessary to provide a matching reference atlas to each histological slice.

2.2 Deep Learning Background

The popularization of deep learning began fairly recently. The known capabilities of these architectures are regularly updated, thus creating motivation to explore different approaches. Additionally, the models are being explored in an experimental way, and it is important to find the most effective techniques by researching previous work.

2.2.1 Deep Learning and Artificial Neural Networks

Deep learning is an area of machine learning that tries to mimic human behavior when interpreting data. A deep learning model learns to understand data by observing a large number of examples and progressively improve its generalizing capacity. In order to understand what deep learning is, it is necessary to first understand what Artificial Neural Networks (ANN) are. An ANN is composed of a set of blocks (neurons), which are organized in layers. The neurons are connected to the other neurons in different layers. A single neuron has the simple task of receiving an input value, incoming either from the data source or a previous neuron's output, multiplying it by the weights, and adding the bias. Each link between two neurons, called an edge, has an associated weight. The result of these two operations is the output of the neuron. Each layer has a different number of neurons. The first layer has one neuron for each value of the data, and the final layer has the number of classes it aims to predict when considering a classification task.

Besides the input and output layers, deep learning models contain layers in between named hidden layers. The number of hidden layers and how many neurons each hidden layer has varied in each architecture. When training a deep learning model, the parameters of the network - weight and bias of each neuron - are constantly being updated and, ideally, progressively getting better at understanding the data.

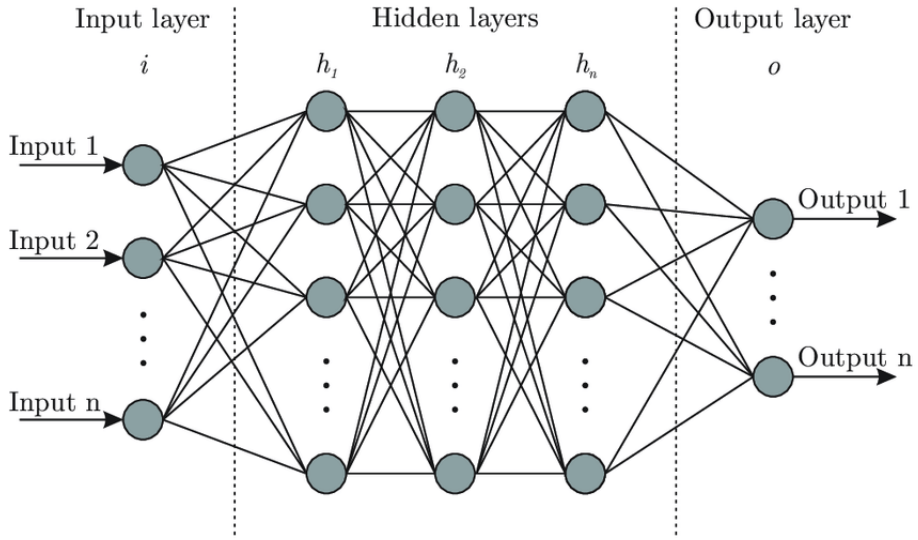


Figure 2.1: Artificial neural network architecture with input, output and hidden layer [11]

When training an ANN, the objective is to improve the predictions by updating the model's parameters. An optimizer is used to minimize the loss of the predictions. There are different types of loss functions, but all of them are used to give the network a sense of how wrong its predictions were. Gradient descent uses the gradients of the loss functions to update the parameters, changing the values slightly. By repeating this step a number of times, the network will get progressively better. The number of repetitions is defined in the model's hyperparameters by selecting the number of epochs. One epoch is one pass through the entire dataset, and it is divided into batches. Instead of optimizing the network with each data instance, the updates are done at the end of each batch.

2.2.2 Convolutional Neural Network

Convolutional Neural Network (CNN) is currently one of the most used and most effective techniques for computer vision tasks. These tasks include image classification, segmentation, recognition, and others. CNNs are a particular type of artificial neural networks that use convolutional layers. It can be disassembled into the basic sections for ease of explanation.

A typical CNN architecture consists of an input layer, a set of convolutional and pooling layers followed by an activation function, a set of fully-connected layers, and a final classification layer dependent on the type of classification. The input layer receives a tensor representative of an image, usually in a square dimension. The convolutional layers are the core component that distinguishes CNNs. They consist of a square matrix of weights that can be seen as a filter to extract information about the image. The pooling layer's function is to reduce the complexity of the data in order to extract information easily. It does so by reducing the feature space. Depending on the type of pooling, it replaces a set of values with a single representative value. Another key component of CNN is the activation function. These are non-linear functions that are crucial for

the network's learning capacity. The final activation layer will give the output of the network, presenting class probability in classification problems or the predicted value in regression problems.

There have been numerous implementations of CNN. We will describe some of the most common architectures used.

- LeNet-5 [32]: It can be considered the standard or base CNN. It was firstly introduced in 1998, and it consists of a five layer network. It has two convolutional layers and three connected layers. It is a simple model that doesn't compete with the state-of-the-art architectures.

- AlexNet [31]: The AlexNet has a similar approach to the LeNet-5 model [32]. However, it added two key aspects fundamental to the modern use of CNN, Rectified Linear Units activation functions and dropout. ReLU functions are commonly used due to their simplicity, and effectiveness [6]. Dropout is a technique that drops some of the units during training in order to deal with overfitting [47]. AlexNet contains eight layers, being five convolutional layers and three fully connected layers. At the ImageNet Challenge of 2012, the AlexNet architecture had the best predictive results [41].

- VGG-16 [44]: Deeper CNNs have been shown to result in higher performance [48] and the authors decided to explore that approach on the CNN architecture. It was built with 16 layers, 13 convolutional, and three fully-connected layers. In order to stack more layers, the model used smaller filters - of size 2x2 and 3x3. However, this network has the disadvantage of taking approximately 500MB of storage space.

- ResNet [23]: Increasing the deepness of CNNs improves accuracy until a certain point. After passing it, it saturates the accuracy, and then it drops. The authors of the ResNet, however, managed to create a model that can have up to 152 layers without losing generalization capacity. The ResNet architecture has some interesting properties such as the use of skip connections, not applying fully connected layers and the use of batch normalization [29]. It is currently one of the most common architectures being used.

- MobileNet [27]: As an answer to the popularity of large models, Howard *et al.* [27] published in 2017 a new architecture named the MobileNet. Like the name suggests this architecture can be used even in mobile devices. Although it may have to compromise in performance, this architecture can be useful in situations where there are memory limitations.

2.2.3 Image Segmentation

While CNNs are often used for image classification, they are not limited to these types of tasks. Image segmentation focuses on processing an image by classifying each pixel into a group - or class - resulting in a clear division of the objects or regions of interest.

In order to overcome the difficulties of performing image segmentation with a traditional CNN, Ronneberger *et al.* introduced an encoder-decoder architecture to more efficiently deal with the segmentation in medical imaging with the name U-Net [40]. The U-Net architecture receives an image as input and it outputs a segmentation mask. In order to train these models, it is necessary to have a segmentation dataset consisted of pairs of images and ground truth masks. The U-Net architecture present by Ronneberger *et al.* is fully-convolutional. Figure 2.2 represents the U-Net's architecture.

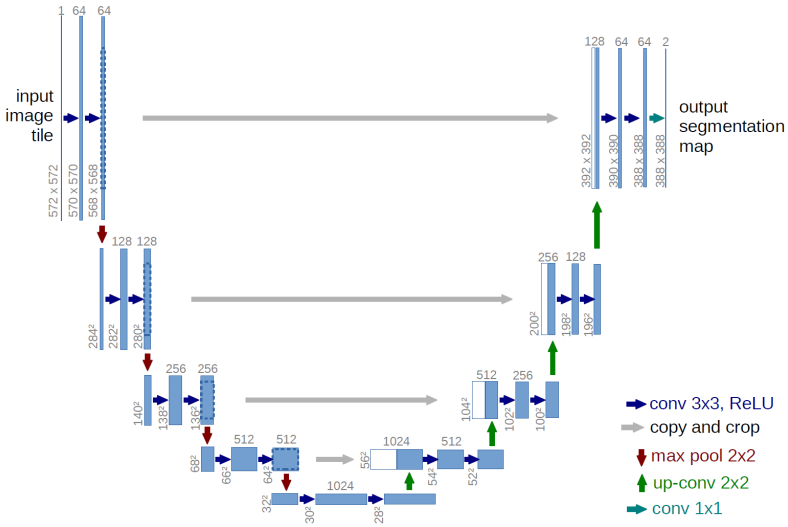


Figure 2.2: U-Net architecture, representing downscalling, upscalling and concatenating steps. Image credit: Ronneberger *et al.* [40]

The encoder component of a U-Net is often replaced by a pre-trained CNN to boost the learning. It downsamples an image through pooling it into an encoded version of its features. The model's architecture is then followed by a decoder section that upsamples the image in order to generate the predicted mask. An important feature of the U-Net architecture compared to other encoder-decoder applications is the use of concatenation of the high resolution features from the encoder to the decoder, maintaining a semantic logic in the upsampling process. The number of channels of the predicted mask varies with the number of segmentation classes. In the original U-Net, the output only contains two channels, making it a binary mask.

2.2.4 Training Procedure Refinements for CNNs

In this final section, we will present a set of "tricks" that have been shown to increase performance and enable faster training in image classification researched by He *et al.* [25].

- Linear scaling learning rate: When the batch size is increased, the noise of the batch is also reduced. Thus we can increase the learning rate to make more significant progress in each batch. The proposed formula to set the learning rate is 0.1 x (Batch Size)/256.

- Learning Rate Warm-up: Since the network's weights are initially random, they are further away from a correct model in the beginning than in the later stages of training. Applying a more aggressive learning rate in an earlier epoch might lead to instabilities. Goyal *et al.* [21] advises to warm-up the learning rate linearly from 0 to the desired initial learning rate. The authors present the following formula: Epoch Number x (Learning Rate)/(Total Number of Warm-up Epochs)

- Zero γ: This technique applies to ResNets [23], and it is used to initially reduce the number of learnable parameters. The ResNet's architecture is built by multiple Residual blocks followed by normalization layers - which can be seen as $\gamma\chi+ \beta$. The authors propose initializing the γ value with 0 for all the layers immediately proceeding the residual blocks.

- No Bias Decay: It is common to apply regularization to both bias and weights. In order to prevent overfitting, it was suggested in [30] to apply regularization only in the weights. Based on this recommendation, the authors suggest applying the decay only to weights in convolutional and fully-connected layers.

- Mixed Precision Training: It was proposed by Micikevicius *et al.* [35] to use FP16 to store parameters, gradients, and activation values while simultaneously keeping an FP32 version of the parameters for updating as a way of speeding up the learning process without losing precision.

- Cosine Learning Rate: Loshchilov *et al.* [33] presents a formula to set the learning rate decay. The authors propose a simpler version that schedules the decay to become 0 at the end of the training. This technique allows for steadier training.

- Label Smoothing: This concept was initially introduced in Inception-v2 [49]. It prevents the model from predicting the labels with very high confidence, which might lead to poor generalization.

- Knowledge Distillation: The technique of knowledge distillation [26] consists in the use of a successfully pre-trained model to teach a student model by copying. A distillation loss is introduced that penalizes the student model based on how different its predictions were when compared to the teacher model.

Combining the "tricks" mentioned, He *et al.* [25] accomplished an improvement from 75.3 (Resnet50 [23]) to 79.29 in the validation accuracy of the top-1 ImageNet classification.

The learning rate is a hyper-parameter with a high influence on the training performance. However, it can be a challenging task to set the correct values. An idea published in an article by Smith [45] and later implemented by the fast.ai library [28], proposed a learning rate finder or a learning rate range test. The goal of this method is to visualize the effects of different learning rates on the loss function. To do this, a test run is performed prior to the training, where the learning rate changes constantly. When plotting the different learning rates and the losses in a typical

scenario, the loss will begin stable. It will then reach a point where the loss starts to descent until the minimum point and from then on the loss will start to highly increase. The learning rate range should be set around the learning rate values that cause the descent phase. These run tests and visualizations can be helpful in grounding the decision of setting this hyper-parameter and be certain that it applies to our model and data.

Chapter 3

State of the Art

Slice-to-image registration has been a topic of interest for the medical community during the last decade, and it motivated researchers to explore the topic. Most of the work done up until recently has not been using deep learning techniques to accomplish their goals [17]. Nevertheless, by observing the recent results obtained with the use of CNNs in computer vision tasks, we believe that a deep learning approach in slice-to-volume integration is a path worth exploring.

Since machine learning techniques are fairly unexplored for this case, we expanded the literature research in two aspects: (1) identifying possible section detection techniques; (2) discovering deep learning uses in other areas that can be brought into our work. We will begin by discussing the first aspect.

3.1 Automatic Medical Image Processing Pipelines

There has been some significant research on automating image registration processes. Our work does not focus on registering itself; however, previous research showed the value and necessity for the automatic pipelines we intend to contribute for.

Niedworok *et al.* [2] developed a mouse brain registration and segmentation tool - aMap - that resulted in human-level accuracy and less variability in registration. The authors discuss the available atlas for mouse brains in the same article, mentioning some of the faced challenges. They argue that the quality of the 3D atlases is still sub-optimal since they either contain a small number of regions or are built from 2D sequences of atlases, which lead to discontinuity of regions. These limitations also restrained the atlas' choice in our project.

In another effort to perform mouse brain registration, Agarwal *et al.* [7] developed a method robust to artifacts. As stated in the article, missing tissue and tears are common artifacts in brain slices. The registration of deformed slices is a necessity for many mouse brain analysis projects and highlights the importance of focusing on artifact robustness in our tool. Contrasting conventional brain sectioning methods, serial two-photon tomography (STPT) is a method that generates a fairly undamaged brain section. The authors used slices obtained from both methods to test their

approach. Brain slices from different extraction techniques can also be used to both develop and test our tool.

3.2 Slice-to-Volume Techniques

Slice-to-volume registration has been mainly explored through dynamic programming; however, with the recent developments in computer vision field, researchers predict that CNN approaches will result in the state of the art results, accomplishing the task faster and more accurately [17]. In this section, we will start by presenting previously used techniques that can be transferred to a deep learning approach. We will then move on to discuss some instances of CNN use in slice-to-volume integration.

Much of the work containing the relevant techniques operate with the goal to perform slice-to-volume registration. Let us start by defining slice-to-volume registration. Ferrante and Paragios [17] defined the topic as a particular instance of image registration - combining two images' data into a single coordinate system - where a two-dimensional input image, usually referred to as a slice, corresponds to a three-dimensional volume. In our work, we will be developing a slice-to-volume integration step that precedes the image registration. This step can be accomplished in different ways. Either through recreating the volume by ordering the slice sequence, by matching each slice to a slice of the reference volume or when using a deep learning model, by predicting the position of the slice directly. While initially reconstructing the volume is a viable method for discovering slice position, this method introduces a banana effect (or z-shift) [5, 34] problem, which is not easy to deal with.

3.2.1 Matching Criterion

When the integration is obtained by matching a histological slice with an atlas, it is necessary to define a fitting matching criterion. Several similarity functions were reviewed, and some conclusions were found. We only considered similarity measures that can be applied to our CNN model.

Despite its simple implementation, the sum of absolute differences and the sum of square difference measures revealed poor results when dealing with noisy images [17]. Since our dataset will contain deformed images and we aim for robustness, it is important to acknowledge this adversity. Furthermore, the mentioned metrics are aimed at monomodal comparisons, and since we are comparing an experimentally obtained image with a reference atlas, our pairing can be considered multimodel.

Dalal and Triggs [15] introduced a feature descriptor (HOG), which allows for a more robust representation of images. It lies on the idea that an object in an image can be represented by the direction of the edges and the distribution of intensity gradients. To do so, an image section is partitioned into small connected regions - referred to as cells. Within each cell, a histogram is created from each pixel's gradient. To improve robustness against light variations, pose, and shadowing, these local histograms are normalized. Each cell belongs to a local block, and the

normalization is done relative to the neighboring cells within the same local block. The blocks are then described by linking the normalized histograms. Since HOG deals with gradient patches instead of single pixels, it is less susceptible to biases while maintaining information about edge direction and strength. Xiong *et al.* [52] chose to use HOG as the similarity measure between the atlas and histological slices. The process starts by aligning the pair through a least-squares estimation method [50] between two point patterns obtained by Shape Context [10]. Even when the two compared slices are not perfectly aligned with HOG, the true gradients are still captured. Also, by normalizing the intensities, the effects of uneven staining are minimized. Prior to the HOG features extraction, the image is smoothed with the Fourier transform to eliminate high-frequency noise and re-scaled to compensate for deformations caused while sectioning. The comparison is ultimately made by summing the squared difference between every block of the two image's HOG feature vectors. In order to decrease sensibility in the similarity measure, large cells are preferred when compiling the HOG features.

Song *et al.* [46] defined a strategy for measuring similarity between the histological and reference slices as a step in their developed software (AMaSiNe) to map specific neurons found on brain slices to the Allen Mouse Reference Atlas [18]. Initially, the experimental images are pre-processed by performing contrast adjustment, edge sharpening, and downsampling to better approximate the reference atlas. The proposed matching algorithm also takes advantage of structural features by using the speed-up-robust-features (SURF) algorithm [9]. The SURF algorithm matches interest points between two objects or images regardless of the position and rotation of these features. The final similarity value is computed by comparing the HOG of the SURF points, creating a similarity measure robust to geometric transformations.

Another option is to use mutual information techniques as a similarity measure in experimentally obtain images. However, this method relies on the initialization state and often leads to improper results, settling for a local minimum [22]. Robust Patch Based Correlation Ratio [37], Self Similarity a-MI [38] and Contextual Conditioned Mutual Information [39] allow for the use of stochastic gradient descent methods to optimize the functions. We will later discuss the CNN-based matching criterion.

Even without considering the reliability of similarity metrics for section detection, the assignment task will be slower and less efficient. Once comparisons can only be performed between two images, without any knowledge of the sections relative other, assigning the correct atlas to each section is a problem of exponential complexity.

3.2.2 Sectioning Angle Inference

Sectioning the brain into slices is a very precise operation. In this thesis, we will be working with coronal slices. Ideally, all the images would show a brain slice matching a plane perpendicular to the sagittal axis. In practice, this is rarely the case, and it is necessary to infer the sectioning angle before performing more complex tasks. In Fig. 3.1, we can see the superior-inferior angle variation.

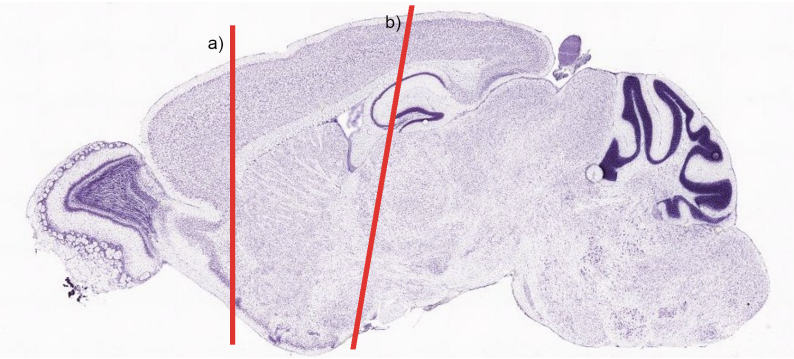


Figure 3.1: Representation of sectioning angle variations in a saggital brain atlas, a) perpendicular angle resulting in a perfect coronal slice b) angle with an error resulting in a superior-inferior tilt in the slice. Image credit: Allen Institute [18]

Xiong *et al.* [52] also dealt with sectioning angle inference. Histological slices are often sectioned with the microtome at consistent angles [19, 53, 20]; thus, the same sectioning angles for every image in a brain sequence can be assumed. Furthermore, it was concluded that the superior-inferior rotation - turning around the frontal axis - tends to be larger due to the irregular shape of the bottom of the brain that causes tilts in the microtome plate during sectioning. Since there is left-right symmetry inherent to the mouse's brain, classifying each side at a time is the same task, thus allowing us to classify them individually. Finally, to infer the angle around the frontal axis, the images were split into upper and lower halves. Their section numbers were predicted individually and combined in order to obtain the rotation with the lowest error. So to determine the actual sectioning angle, each histological slice is initially matched with the best coronal atlas without considering any tilt. Considering that slices are parallel, each matched atlases is introduced to the same angle variation, and the similarities between the pairs are recalculated. The angle dynamically varies until it reaches the value with the lowest total cost. Considering the same angle throughout the whole brain is likely to be the only viable method using dynamic programming strategies. However, depending on the slice acquisition method, this assumption can be incorrect. As it was seen in the brain slices collected by Medizinische Universitat Innsbruck [1], the angle may vary in the same brain due to tissue deformations and movements caused by the cutting step.

Similarly, Song *et al.* [46] approached the angle detection issue by comparing histological slices with differently oriented atlases. Using their matching criterion explained in Section 3.2.1, the image and Allen 3D Reference Atlas [18] similarity is calculated, and the chosen parameters - position and angles - are the ones of the atlas with the best matching score.

3.2.3 Similar CNN Work

Despite being less common, deep learning approaches have been previously researched. Deep learning-based slice-to-volume integration methods were estimated to be faster and more accurate

by Ferrante and Paragios [17]. We will describe in this section CNN approaches and uses within slice-to-volume integration.

Almogadwy *et al.* [8] introduced a CNN to predict slice position in an embryo reference model. The process starts by generating the training data. In order to do so, the selected 3D reference model is sectioned at different angles and positions. There were 515 positions and 90 angles considered, leading to a total of 92700 images in the dataset. Since the data is generated from a reference model, the images do not contain any deformations, which simplifies the prediction process, limiting the predictive capacity to controlled datasets rarely found in experimental contexts. The images were pre-processed, resulting in 128 by 128 pixels grayscaled and normalized images labeled by the pair position-angle. To further increase the dataset, data augmentation was introduced by rotation, skewing, and shifting the images. Due to the low variations between neighboring atlases, the similarity measure was studied by comparing the image to multiple images, presenting some insight into the compromise of the number of classes. It was found that comparing an image with a single distance has a 98% degree of similarity, while at ten units, it drops to 82%. The variations seen showed that while using larger distances can reduce the number of classes and improving the model, it simultaneously reduces the accuracy when distances are too large. The CNN classified the images in three different ways: considering all distance measures, every fifth distance unit, and every tenth. It was concluded that the accuracy is maximized when using increments of ten, resulting in a lower number of classes. The CNN's architecture contains five convolutional layers, a normalization layer, and three fully connected layers, with normalization, max pooling, and dropout layers immediately after each convolutional layer and ReLU activation functions.

Chen *et al.* [12] developed the SCSVRnet for brain slice-to-volume registration. It focuses on solving a series of complex issues such as real-time classification, fully automatic and initialization independent processing, and multimodel and landmark free capacity. It utilizes silhouettes to aid the multimodel matching. A silhouette extraction step was added to the network, followed by spindle transformation to reduce the dimension space. Finally, a prediction step is used to infer the positioning.

Deep learning was also used with a different approach. Simonovsky *et al.* [43] explored CNN as a matching criterion in multimodal medical imaging. It is developed as a classification task for determining aligned or misaligned multimodal pairs. The CNN predicts the disparity between two cubic patches. The architecture is based on a two-channel network introduced by Zagoruyko and Komodakis [54] for image patch comparison. Pseudo-siamese networks introduced [54] were also explored and lead to a decrease in quality. To obtain the training data, patches of the multimodal pairs are extracted. Each patch has the same dimensions, although they suffer positioning, scaling, rotation, and mirroring transformations. This data augmentation is performed online, leading to a practically endless dataset. When the transformations on each side of the pair are the same, the two patches are labeled as being similar; otherwise, they are labeled as dissimilar. The network aims at minimizing the hinge loss with an SGD optimizer. The results showed a significant improvement over mutual information methods.

To the best of our knowledge, there are no public labeled datasets for mouse brain coronal slice detection with experimental images. There is, however, a large number of brain sections obtained with a serial two-photon tomography in Allen's [18] transgenic line datasets. While these images do not have section numbers associated, they are presented sequentially, which makes manual labelling a simpler and more viable process. However, since the brain slices are obtained using a different method, the quality of the sections is better, containing a low amount of deformations. Such images are helpful due to their clarity but can not be exclusively used since the most common slice images capturing methods result in frequent deformations.

3.3 Summary

Throughout this section, we reviewed prior research, and it will be a base for our work. Regarding the specific case of slice-to-volume integration, it can be concluded that dynamic programming techniques are the predominant approach, but in the later years, deep learning has marked an important step in the field. Both types of techniques will be relevant to our work since they either provide inspiration or useful functionalities for our case. Table 3.1 presents an overview of the mentioned papers discussed in this chapter. In the next chapters, we will discuss our approaches to the pairing problem.

Authors	Data type	CNN-based	Similarity Measue	Angle Detection
Xiong *et al.*[52]	Coronal Mouse Brain Slices	No	HOG differences	Atlas Similarity
Song *et al.*[46]	Coronal Mouse Brain Slices	No	HOG and SURF points differences	Atlas Similarity
Almogadwy *et al.*[8]	Synthetic Embryo Slices	Yes	-	Predicted
Chen *et al.*[12]	Multimodal Brain Images	Yes	-	Predicted
Simonovsky *et al.*[43]	Brain MRI	Yes	CNN metric	-

Table 3.1: Overview of state of the art

Chapter 4

Section Detection Specification

Throughout this chapter, we will further define the challenge of section detection by describing the main task and its inherited obstacles. We will, as well, describe the available data provided by the Medizinische Universität Innsbruck [1]. Finally, we will present and justify our proposed pipeline for dealing with mouse brain section detection.

4.1 Problem Specification

Section detection, at its core, consists of correctly inferring a coronal brain slice's position in the anterior-posterior axis. We will refer to the coronal mouse brain slices as sections.

Considering that a brain volume comprises multiple three-dimensional regions, to analyze a single brain section correctly, it is necessary to visualize the 2D projections of said regions in the section's plane. To simplify both the visualization and specifications of brain regions into two-dimensional planes, a convention was established to divide a brain model into 132 sections. The Allen Institute for Brain Science [18] uses this convection in its Mouse Brain Atlas, which will be used in our section detection solution. This institution provides both 3D mouse brain models and atlas sequences for each one of the 132 sections. The 132 reference atlases are uniformly separated by 100 µm intervals, increasing from anterior (or frontal) to posterior. The brain regions in each reference atlas are defined in a hierarchical structure, allowing the visualization of regions and sub-regions at different levels of deepness. The reference atlas hierarchical structure will be further clarified in Chapter 6, where we will discuss atlas generation.

In our approach, we will be using a dataset built on sections labeled by clinical experts. The labeling is done by comparing the features of an experimentally obtained section with the features of the atlases. To do so, the clinical technicians recur to the 132 2D atlases provided by the Allen Institute. The atlas with the largest similarly is chosen, and the experimental section is attributed the number of that same atlas, obtaining a number for each section identifying its position. The

goal of section detection is to automatically attribute the correct number for each section, specifying which atlas best represents the present regions in the sections.

An ideal coronal section would be extracted perpendicular to the anterior-posterior axis. However, it is not unusual to present an undesired angle to the cutting plane, resulting in tilted coronal sections. These tilted sections can not immediately be assumed to be comparable to the perfectly coronal reference atlas since the tilt may have introduced brain regions from neighboring sections. To minimize the error from passing to further steps, the section detection solution must incorporate an angled reference atlas to match the experimental sections adequately. It must be noticed that the angle variances can occur on two different axes. Let us consider a plane with a horizontal and a vertical axis, named x and y, respectively. A rotation over the y-axis will cause a tilt between the left and right semi-planes, while a rotation over the x-axis will introduce a tilt between the upper and lower semi-planes. The tilts will be referred to as an α-rotation and β-rotation, respectively.

Assuming the possibility of an inconsistent cutting angle, a single number per section was considered insufficient since it inherently assumes a coronal cut. In discussion with the clinical department responsible for data labeling, we decided to proceed with a new labeling system that considers the cutting tilt. Contrary to the previous method, each section would be divided into multiple parts, and a reference atlas number would be attributed to each one. The data labeling system will be further presented in the next subsection.

Besides the variation of the cutting angles, there is another factor that may demand changes to the coronal reference atlas before attributing it to an experimental section. When considering a single section with a number between 84 and 103 out of the 132 possible positions, we can often observe that the three present regions - left cerebrum, right cerebrum and, brain stem - are no longer connected. The tissue that connects these three regions is fragile at this point, and it is common to break, leaving space between them. Due to the anatomical properties of the brain stem, when it is disconnected from the cerebrum, it is vulnerable to movement during the cutting step, causing it to rise or sink in relation to the cerebrum. It is necessary to consider that the regions in these sections may not belong to the same reference atlas. To correctly attribute an atlas to the brain section, we once again must be able to identify multiple atlas numbers within a single section.

Moreover, regarding the sections positioned between numbers 84 and 103, we are faced with another problem. Since the section can contain unconnected parts, it also happens that some of the regions are lost in the cutting process. Figures 4.1, 4.2 and 4.3 show how missing regions can affect the appearance of these sections. Identifying these incomplete sections is needed to be able to attribute the atlas correctly and to perform atlas to section registration later. The brain stem must be present to consider it a region, resulting in four different possibilities. The sections can contain all the regions, it can contain only the left or only the right cerebrum, or it can only show the brain stem, with both the cerebrum regions excluded.

After describing the previous problems with attributing a single coronal reference atlas to an experimental section, we can conclude that using the 3D mouse brain models provided by the Allen Institute is more beneficial than the 132 two-dimensional reference atlases. Our section

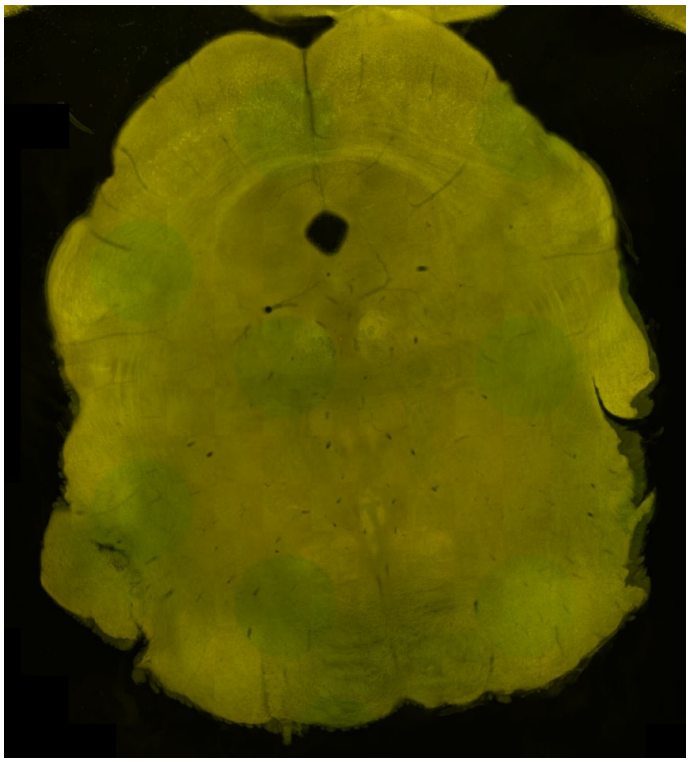


Figure 4.1: Example of section number 98 containing only brain stem

detection solution must be accompanied by an atlas generation step. The atlas extraction must be flexible to accommodate the angled cuts and off-plane regions. By using a brain reference volume, an extraction algorithm must be developed to fit the requirements and replace the 2D atlas sequence used by the clinical team in the labeling process.

One more obstacle must be considered in the development of our solution. Similarly to the missing region's issue described previously, other deformations in the brain sections can be found. We refer to these deformations as artifacts. The artifacts can be missing tissues, holes, tears, air bubbles, folded tissue, and uneven staining. These common issues make the section detection step more difficult, and it forces us to focus on robust approaches. We consider approaches based on similarity functions insufficient to deal with regular and significant artifacts and decided to explore a deep learning method for section detection.

Lastly, it is important to notice that not all sections will have the same registration method. For the sections with numbers below or equal to 22, from a clinical perspective, it is only necessary to analyze very few regions. Since these regions are quite evident in the experimental slices, an image segmentation method is enough to perform the registration. Consequently, it is not necessary to infer the angle nor extract an atlas for these sections. For this reason, the section detection step should also be responsible for pre-selecting the sections and pass them into the correct image registration processes.

Our solution consists of dividing each section into the relevant quadrants and classifying each

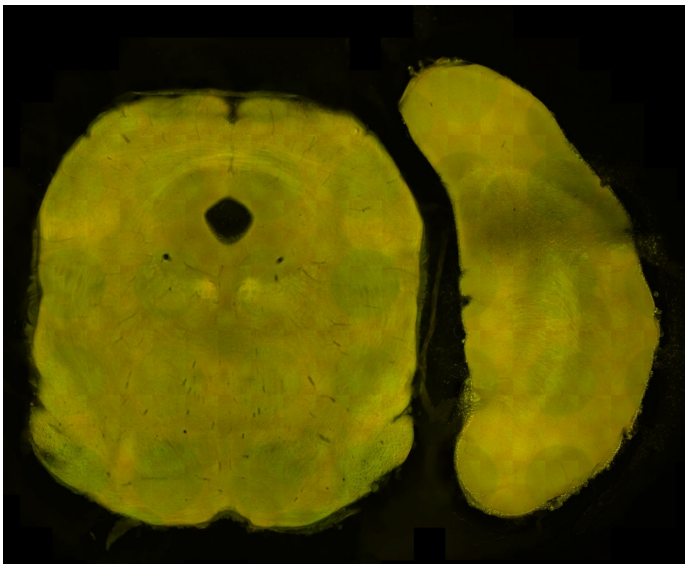


Figure 4.2: Example of section number 97 containing the brain stem and right cerebrum

quadrant individually. The extracted quadrants are different in sections between 84 and 103. The assigned section numbers for each quadrant are then combined and a tilted atlas can be generated. Figure 4.4 represents our method. In Section 4.3 we will further describe our proposed pipeline.

4.2 Available Data

The available data was labeled by the Medizinische Universität Innsbruck [1]. We can consider that three different datasets constitute the available data for our work. We will refer to them as Dataset 1 (DS1), Dataset 2 (DS2), and Dataset 3 (DS3). While DS1 and DS2 were also collected by the Medizinische Universität Innsbruck [1], DS3 was available as transgenic line datasets in the Allen Institute of Brain Science repository [18]. In the early stages of our work, only DS1 was ready to be used. Upon experimenting and discussing the details of section detection mentioned in the previous section, we establish a new convention for labeling the DS2. DS3 was later introduced, since we noticed that the available detailed data (DS2) would be too little to obtain the best results. Thus some of Allen's transgenic line datasets were categorized into our labeling system and could be used as well. While similar to the other dataset's images, the data in DS3 was obtained using a different method than the one used by the Medizinische Universität Innsbruck [1]. The Dataset 3 images were acquired using serial two-photon tomography, which resulted in brain sections with a lower amount of artifacts.

The most evident difference between the datasets is the number of labels per image. The first dataset was constructed without considering a multiple-label system. Each section image contains a single label that corresponds to the best fitting atlas. DS1 has 1655 labeled coronal slices. Dataset 2 is significantly smaller than Dataset 1, containing 399 images. Dataset 3 then introduced 2934 new images, which were labeled similarly to Dataset 2.

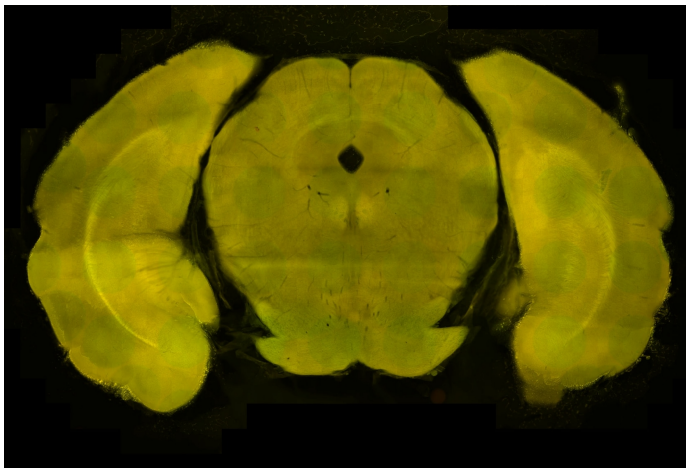


Figure 4.3: Example of section number 95 without missing regions

There is some unbalance in the class distribution of DS1. Looking at the Figure 4.5, we can observe a spike in categories 17, 18, and 19. When observing sections with numbers below 20, it was considered that there was very little differentiation between neighboring slices, thus facing many challenges classifying them. Taking into consideration that these sections will not advance to the atlas registration step, it was decided to group all sections below 20 with the labels 17, 18, and 19. Label 17 includes section numbers 1 to 4, label 18 includes the section numbers 5 to 15 and, label includes the section numbers 16 to 19. This reduced the sparsity of the data without relevant drawbacks.

The labeling system in DS2 and DS3 includes different labels for a single section. We will refer to this as quadrant labeling. With the quadrant labeling system, we split the image into different quadrants, classifying each one individually and calculating the cutting plane for each section.

Figures 4.6 and 4.7 represent the number of examples in each class. Once more, in DS3 it is visible the increase in section numbers 17, 18, and 19 due to our conversion method for Group B sections previously explained.

4.2.1 Quadrant Labelling System

The quadrant labeling is not equal in all the sections. Depending on their anterior-posterior position and consequently their section numbers, the sections in DS2 are categorized into three different groups: Group A, Group B, and Group C.

4.2.1.1 Group A

Group A incorporates the largest amount of region numbers. Here we consider all the sections with numbers from 23 to 83 and 104 to 132, inclusively. Sections within this group are divided into four equal quadrants, and each one is labeled individually. The quadrants are obtained by considering

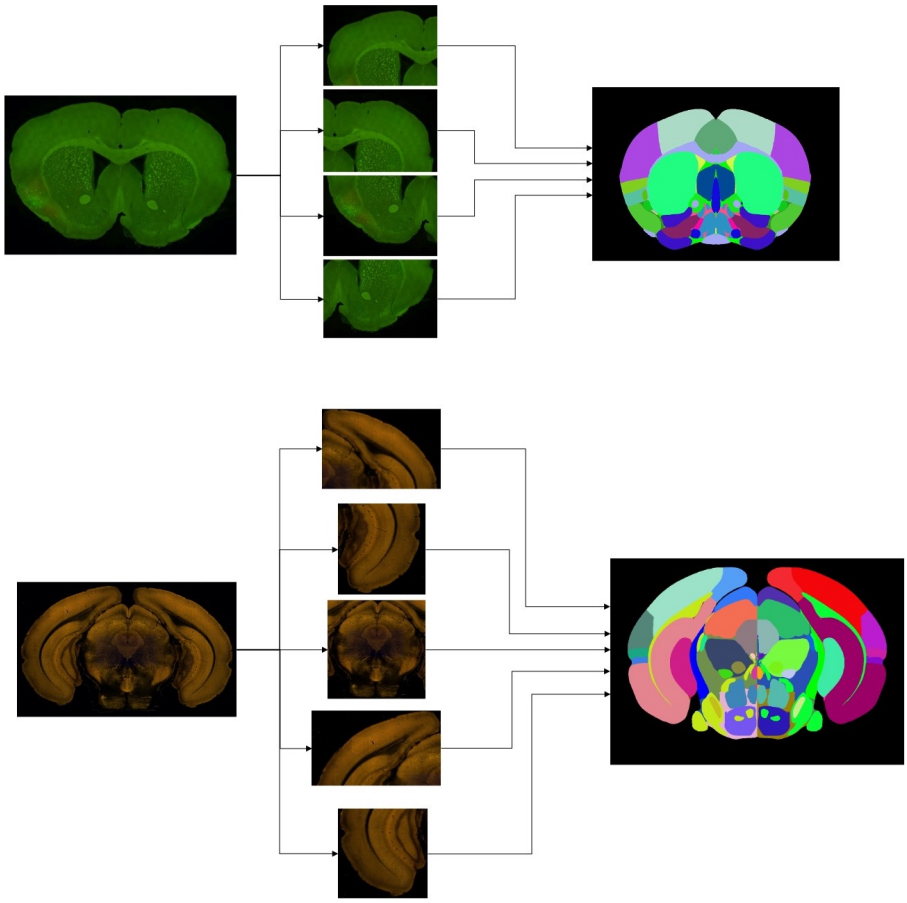


Figure 4.4: Quadrant splitting and atlas generation in different sections. Most sections are split into four quadrants (top) while sections between 84 and 102 are split into five quadrants (bottom) to generate the appropriate atlas

two centered lines, one vertical and one horizontal. The quadrants are fit the following naming system (see Figure 4.8):

- Q1: Left-upper quadrant
- Q2: Left-lower quadrant
- Q3: Right-upper quadrant
- Q4: Right-lower quadrant

With the four labels, it is possible to infer both the α-rotation and the β-rotation. Through this method, we can utilize the characteristics visible to the clinical team to discover the cutting angle and, consequently, generate an ideal reference atlas.

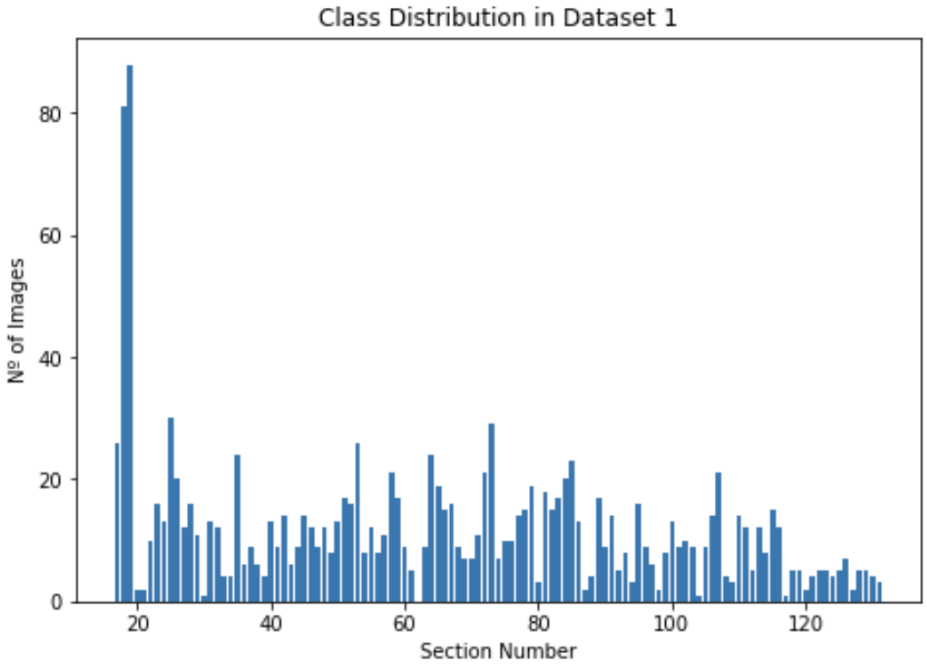


Figure 4.5: Data distribution throughout the classes in Dataset 1

4.2.1.2 Group B

In this group, we consider the sections with numbers below or equal to 22. These sections are registered in an image segmentation step, and it is not necessary to generate a reference atlas for these sections. To appropriately classify the frontal sections, these must only be classified as Group A sections; thus, we only consider one single label, similarly to the labeling system in DS1, approximating their position number and disregarding any tilts in the cutting plane.

4.2.1.3 Group C

All the sections with numbers between 84 and 103 are contained in Group C. As it was previously mentioned in the problem specification, some of the regions in these sections are often separated from the rest of the slice, which can cause the brain stem to be in a different cutting plane. To deal with these deformations, a new value was added to the quadrant labeling system used for Group A. The new label, named Q5, refers to the atlas number of the brain stem, which is located at the center of the section. The remaining labels (Q1 to Q4) are applied to the cerebrum regions (see Figure 4.9).

4.2.2 Data Deformations

Particularly Dataset 1 and Dataset 2 contain deformations that must be taken into account. To build a robust model capable of mining information from these datasets, it is necessary to identify any regular variations or issues in the images. It is quite evident that the contrast of the images

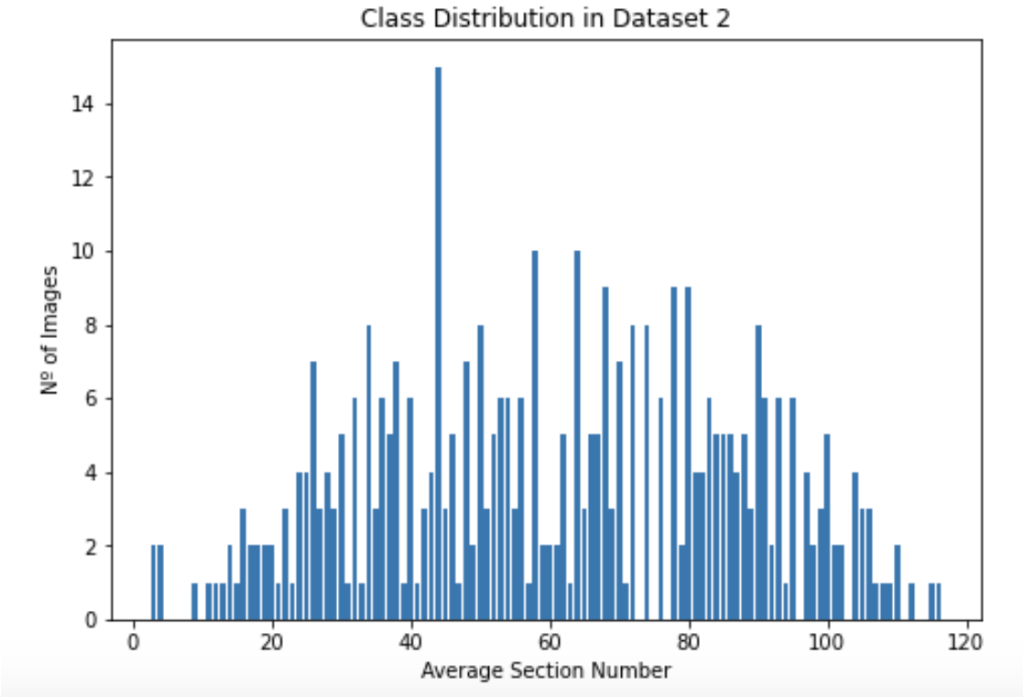


Figure 4.6: Data distribution throughout the classes in Dataset 2

varies significantly throughout the dataset. The brain slice images are obtained by adding a slide containing up to 10 brain sections into a high-resolution scanner. Some differences between the brain slice cause the scanner to extract images with very different lighting conditions, resulting in a varying image contrast throughout the data. While manual contrast correction is possible, it is important to build a solution capable of automatically dealing with the variations. Similarly to the contrast differences, the coloring of the sections also changes regularly. Images are seen with yellow, green or red tonalities, with a wide range of intensities. However, the different coloring does not reflect any relevant meaning for our work, and it should be ignored by our tool.

The brain slices are carefully placed into the slide to be scanned, ideally without any rotation and with enough distance to avoid overlapping of two different slices. Nevertheless, some unwanted movements can occur, and both these deformations are introduced to the dataset. Even though both the rotations and overlapping are usually small when present, it is still worth noticing as possible obstacles in our proposed method.

Perhaps the most difficult deformations to deal with are the artifacts mentioned in the previous section. Artifacts are very often found in slice images, and they can appear in very different forms. In Figure 4.11 a) we can see an example of a section containing many artifacts. It contains tissue tears in the left and bottom parts, with some missing tissue in the bottom. Furthermore, we can observe luminous round spots on the lower left side of the slice. Meanwhile, Figure 4.11 b) represents a slice with the same section number as Figure 4.11 a) but with fewer artifacts. Despite their strong visual difference, both sections should be categorized equally.

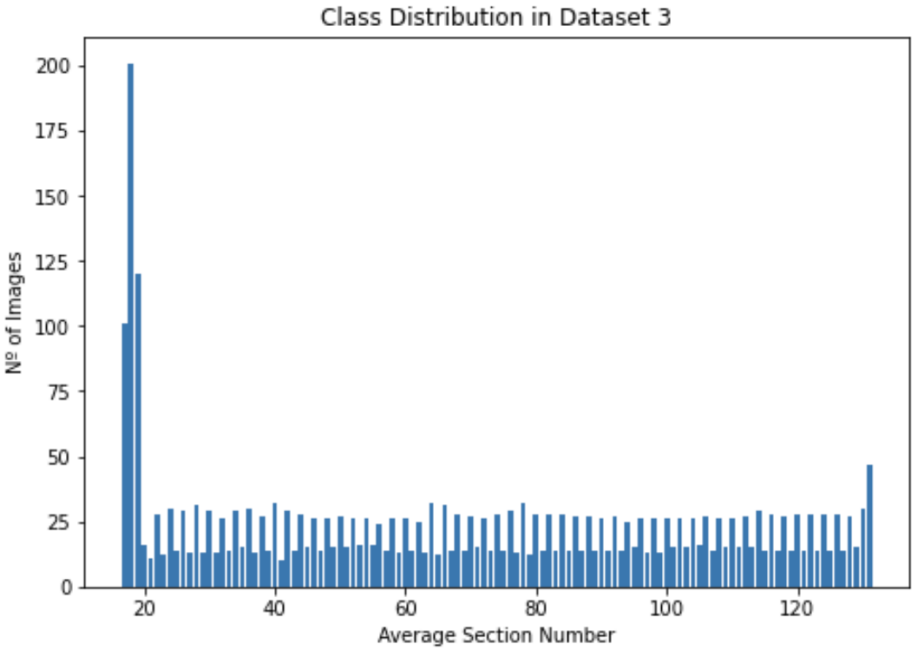


Figure 4.7: Data distribution throughout the classes in Dataset 3

The artifacts can even be more serious, causing significant pieces of the section to disappear. This is represented in Figure 4.11 c), where the top-right quarter of the image is missing, and in Figure 4.11 d) where the whole right side of the image is partly or totally hidden.

Regardless of the present deformations, the images shown must be processed by our system and correctly categorized. Our focus will be on creating a robust solution where sections can be identified only by some specific features for each section number.

4.3 Proposed Pipeline

Upon understanding the full requisites the section detection tool must fulfill, it was possible to specify a pipeline to process a set of brain section images.

Acknowledging that different slices will have different registration processes, the first step of the pipeline is to introduce a selection step capable of distinguishing the Group B sections from the remaining. Afterwards, these frontal sections will be directed to the correspondent registration processes. Since the registration process will not be integrated into our work, after identifying them, we will discard Group B sections from our pipeline.

Performing automatic quadrant labeling is a crucial aspect of our solution. We propose to perform this step by splitting a full image into the quadrants, obtaining a different image for each quadrant. Each of these images will then be classified individually, simplifying the classification problem from multi-label images to single-label images.

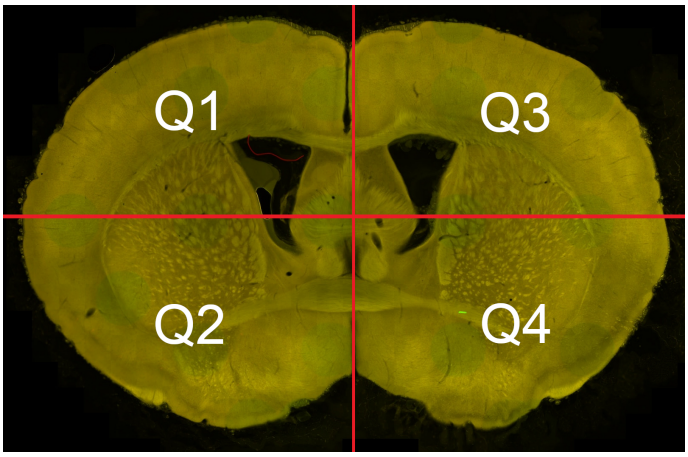


Figure 4.8: Quadrant distribution in Group A sections

Since Group A and Group C have different quadrant types and labeling systems, splitting images from different groups in quadrants is also done differently. With Group A, we can obtain the quadrant images by cutting the image in half both vertically and horizontally. In Group C images, we first must identify the three present regions: left and right cerebrum and brain stem (see Figure 4.12). The cerebrum quadrant will be extracted by splitting them into top and bottom halves, while the Q5 quadrant will be the full brain stem region.

In order to properly perform the image splitting step, the first selection should also account for Group A and Group C differentiation. Due to the high presence of artifacts, the challenging nature of the problem, and the amount of data available, we will focus on deep learning approaches. Initially, it was considered to develop a classifier aimed at predicting for each image the correct group it belonged to. However, the same results could be obtained by predicting the section number of each image and fit them into the groups based on the value obtained. The latter method was chosen since it did not show significant sacrifices, and it allows us to obtain useful information for later steps.

Once the correct groups are defined for each slice, the splitting algorithm extracts the quadrants correctly and passes them to a new model. This model must predict the correct section number for the quadrant. Once all the quadrants are assigned a value, the cutting plane can be calculated, obtaining the section number, α-rotation, and β-rotation. These values are passed to the atlas generation step, where an appropriate reference atlas will be extracted from the 3D brain model. Finally, the section image and the extraction atlas are passed to the image registration phase. Figure 4.13 represents the pipeline we propose to solve section detection.

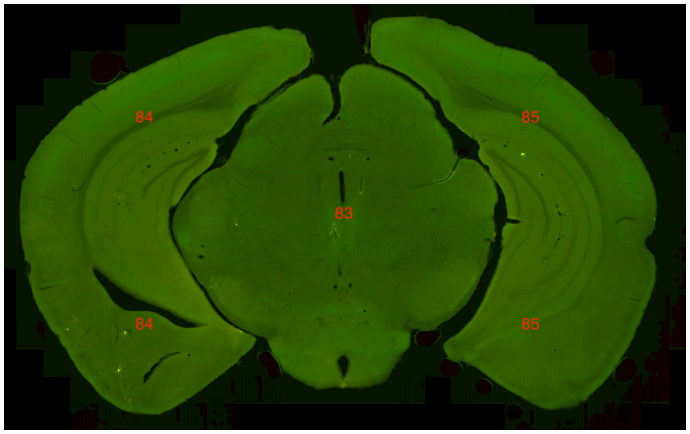


Figure 4.9: Different values in quadrant distribution in Group C sections, showing an off-plane brain stem with a Q5 of 83, while the cerebrum regions suggest an α-rotation

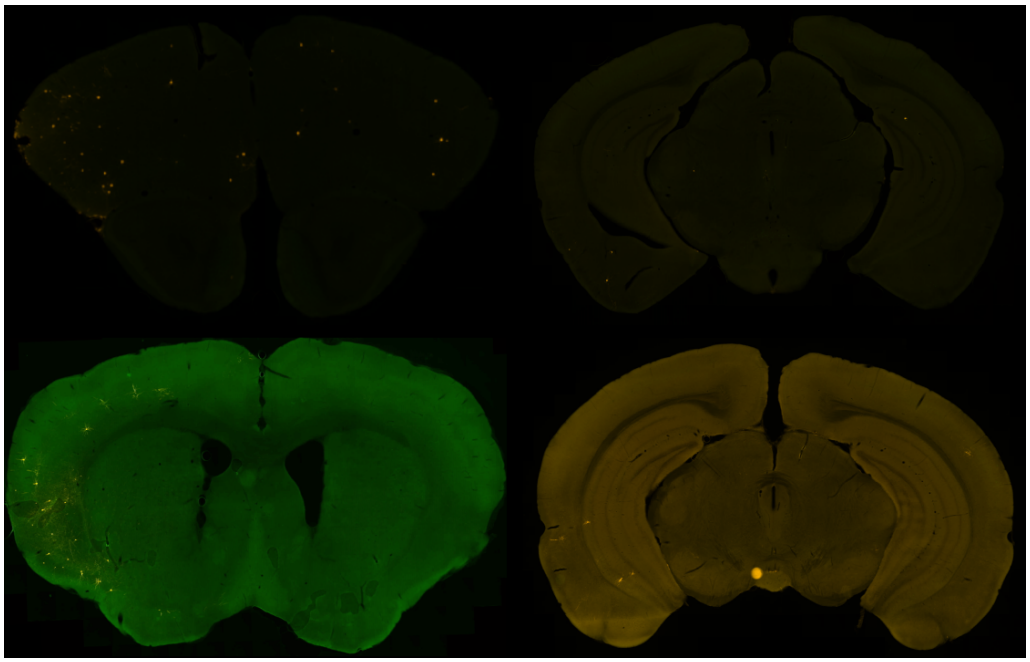


Figure 4.10: Contrast and color variations in brain slice images. Sections can have poor light conditions, making it challenging to analyze them without applying contrast corrections (top)

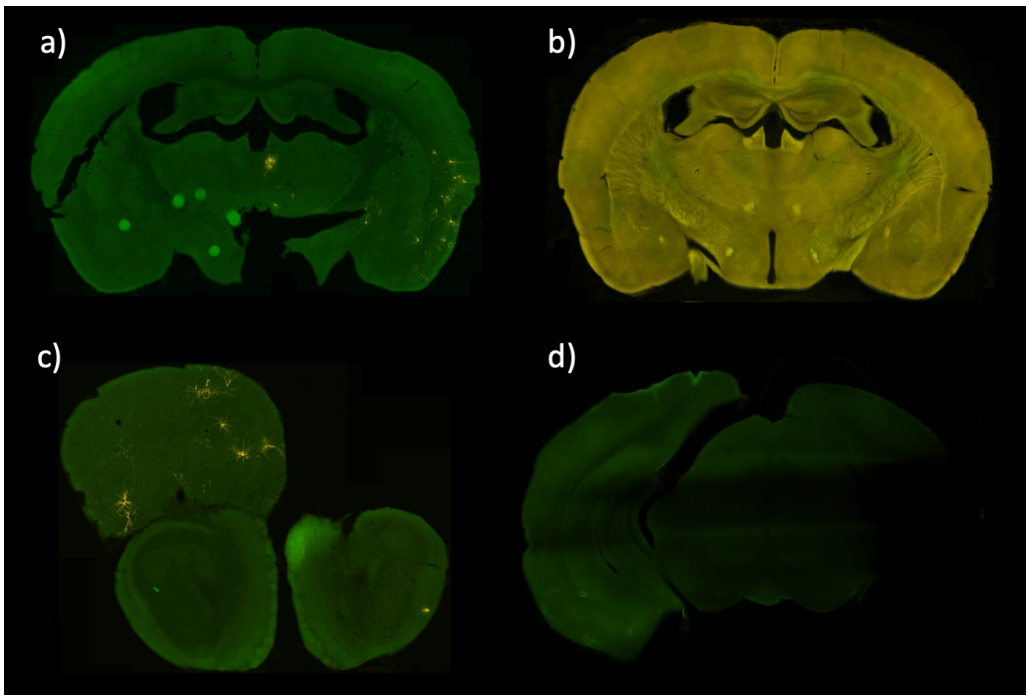


Figure 4.11: a) Section number 66 example containing significant tears, missing tissue and luminous spots b) Section number 66 example containing little artifacts c) Section number 26 example with missing region on the top-right quadrant (Q3) d) Section number 85 example with faded regions in the right quadrants

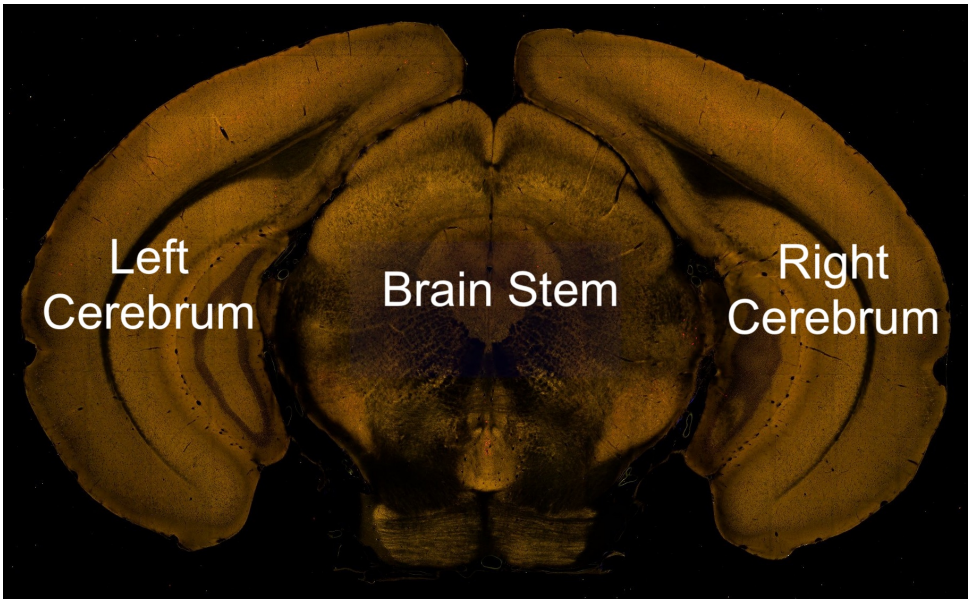


Figure 4.12: Regions of a Group C section

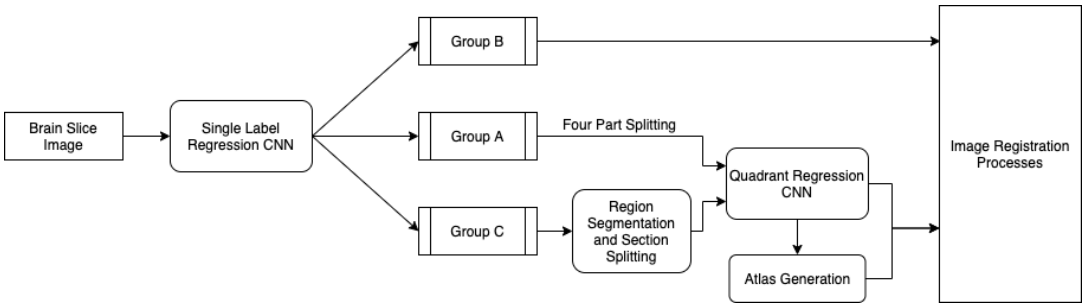


Figure 4.13: Representation of the proposed pipeline

Chapter 5

CNN-based Section Detection

Our section detection method heavily relies on convolutional neural networks to accurately predict the slice position in the brain. There are three deep learning models in our pipeline, all of them developed using TensorFlow [4] and Keras API [13].

5.1 Single Label Classifier and Quadrant Label Classifier

The Single Label Classifier and Quadrant Label Classifier are similar models and there were many shared methods between the two. In this section, we describe the two deep learning models.

5.1.1 Single Label Classifier Overview

We named the first model the Single Label Classifier. In this step, the input slice images are assigned section numbers that reflect their coronal position in the brain. This information will mainly be used to classify the brain section into the correct group, although it will also assist in the atlas generation step.

Since the main goal of this model is to classify a slice image into one of the three groups - Group A, Group B, or Group C - a three-label classifier was considered. However, the three-label classifier was discarded, and a regression model which predicts the section number was used instead. This decision was justified for several reasons. Using a model that predicts the section number of the slice will give more information about the image than simply fitting it into a group. Using the predicted section number, we can then assign the slice to the correct group while still having an approximate section number. Furthermore, we did not experience any difference in the group assignment results when comparing the three-part classifier with the regression. Having the section number prediction allowed not only the later use of this value but also facilitated the development of the project. While the necessity for a group classifier was clear in the early stages, the exact borders which limited the groups were prone to changes throughout the development. In the case of the three group classifier, any change to the group borders would require re-training

the model, as opposed to the regression model, where the same model would allow different group borders since the classification would be posterior to the section number prediction. The three group classifier allowed, however, the possibility to use smaller models that would not take as much memory. Nonetheless, it was possible to develop a number prediction model of acceptable size, and it was decided that the slight increase in memory would be worth the advantages.

The single label classifier must attribute the correct label to each brain slice. While these labels represent different atlases or region distribution in the section images, there is continuity between labels. In some cases, the difference between two consecutive atlases is not visible. The regions and the section shapes change fluidly, which can make the binary distinction between two consecutive section numbers a difficult trait for the model to learn. While a regression model would look for continuity in the labels, a classification model would not focus on the continuity of the section numbers and would only consider rigid labels. For these reasons, we decided to focus on implementing a regression model over a classification model.

All the three datasets available could be used to train the single-label deep learning model. For Dataset 1, the section number labels were used, and for Datasets 2 and 3, the multiple labels are averaged to obtain a single number position. A selection of 246 images was separated from Dataset 1 to be used as a test set. The test set contains images from every class and a significant amount of artifacts present to understand the results better and analyze them with a higher degree of confidence. With the selected test images excluded, the datasets were combined, resulting in 4778 brain slice images. From this training dataset, 20 percent of the data was randomly selected and separated to be used as validation.

5.1.2 Quadrant Label Classifier Overview

The Quadrant Label Classifier has a high degree of similarity with the Single Label classifier. Thus, many of the developed methods of data processing, data augmentation, model analysis, and model architectures are common to both neural networks. Since the Single Label Classifier was developed before, much of the knowledge and understanding of the problem was applied in the Quadrant Label Classifier, which simplified the development and required less experimentation. For the same reasons explained for the Single Label Classifier, the quadrant prediction model was also implemented as a regression.

This task has some added difficulties when compared to the first classifier's task. First of all, the quadrants only contain a small portion of the brain section and, consequently, a lower amount of identifiable features per image. Then, when considering quadrants, there is a big difference between images of the same class. This is caused by the superior-inferior division of the brain. The section images have a left-right symmetry which means that features are shared between left quadrants and right quadrants. However, since there is no top-bottom symmetry, the identifying features of a section number are generally different from top quadrants to bottom quadrants. Furthermore, the central quadrant present in the Group C sections will also contain different features. Despite the diversity, the CNN must be capable of attributing the same section numbers to two or

three different types of images. To fight this issue, it was initially considered to split the quadrant label classifier into two CNNs, using one for superior quadrants and the other for inferior quadrants. This idea was later discarded when analyzing the results of a general quadrant labeling CNN. While it required a denser architecture - with more layers and parameters - it seemed to be able to distinguish and separate the different quadrant types accurately. It is important to consider that while the task had bigger challenges since the data consists of section quadrants, the amount of data is approximately four times larger than the original unsplit data, which also significantly improves the learning capacity.

5.1.2.1 Data Preparation

Due to the different labeling systems, not all the available datasets could be used for this model. In particular, Dataset 1 only contains a single label per image, not passing any information about cutting angle or quadrant differences. Mainly due to the small amount of available data in the early stages of the project, it was still considered to divide Dataset 1 into quadrants and use the single section number as a label for each quadrant. Since this would only be an approximation, it could only be used as a pre-train step and had to be later tuned with the accurate data. While theoretically viable, this was also deemed unnecessary once we managed to increase the amount of data in Dataset 3.

By combining DS2 and DS3, 3333 full section images were available for quadrant classification. This data had to be divided into quadrants before it was usable for training. Since quadrant labeling only applies to Group A and C images, it was first necessary to filter the Group B sections. As a redundant step and to deal with errors coming from the single-label classifier, the higher section numbers of the Group B sections were also included in the data. By adding this step, if a high section number from Group B is classified as Group A, it will still be assigned with an atlas, and the error can be eliminated. The Group C sections also had to be identified since the quadrant splitting is done differently. To separate these sections, our quadrant data generator algorithm looks for the Q5 value. If the Q5 field is empty, it is not considered to be a Group C section.

For each Group A image, it is important to understand which quadrants are present in the image. Group A splitting is done by simply dividing the image by its dimensions. Initially, the quadrants were taken by splitting the image in half both horizontally and vertically, obtaining a patch with half the width and half the height of the original image. We then increased the quadrant space by 10%, meaning that the dimensions of a quadrant will be 60% of the image's height and width. The quadrants will have some overlapping information; however, we do not consider this to cause any issues. The artifacts can cause quadrants to disappear from the image, and this information is reflected in the labeling system. If the quadrant is missing, the value of the respective field will be null. With this feature, after dividing the full image into four parts, each quadrant's value is checked, and when a null value is encountered, the quadrant is not saved into the split dataset. The images from DS3 did not have any missing regions; thus, all the quadrants were always saved.

Splitting Group C sections was a more complicated process than with Group A. This was mainly due to the common tilts in these sections and significant differences between the same regions in different images. Additionally, as previously explained, missing regions in these sections are likely to occur. While we could fix the missing region problem by using the null values as a way of identifying the unwanted quadrants, two problems remained: there were no patterns of where to cut the image into quadrants, and this method could not be used when using the CNN in production, since labels would be required. Thus, a segmentation CNN was introduced, which allowed for quadrant generation in Group C images. The specification of this model will be presented in Section 5.2.

The train dataset consisted of 10462 images quadrant images, in which 8369 were used for train and 2093 for validation. Additionally, two full mouse brains were used as a test dataset, which contained 627 images.

5.1.3 Data Pre-Processing and Augmentation

To both simplify and help the model generalize during training, each image was transformed before entering into the model. The first step in the pre-processing is to resize the images to the correct dimensions. The appropriate image dimensions in deep learning need to be a compromise between image quality and image size. While a high-resolution image contains more details and possibly more features, it can also lower the performance of the model when considering training and prediction times. Furthermore, a higher detail might not be of any use to the image since the identifying features may be visible at a lower resolution, and high-level details may point to irrelevant characteristics. The dimensions chosen were 200 by 200 pixels, creating a square image with enough identifying details while keeping an affordable size. The following step is to transform the colored images from RGB to grayscale. Scanned brain slice images often appear with different colors; however, this color difference does not have any relevant meaning to our task, and thus it can be eliminated by removing insignificant attributes such as different coloring or contrast variations, the data similarity increases, which will force the model to learn the defining class characteristics and not focus on irrelevant traits.

Due to the scanning methods used, the contrast in the images varies significantly, particularly in datasets 1 and 2. Furthermore, highlighted neurons and certain artifacts create very bright spots in some slices. These characteristics of the data created the necessity to develop a normalization function that would fix the contrast difference while not being affected by minuscule bright spots in the images. To facilitate the use of the images, the image values were decided to be limited between 0 and 1. While dividing all the values by 255 would fit the image into our 0 to 1 limit criterion, it does not contribute to contrast normalization. Another type of normalization was then considered, where all the values in the image were subtracted by the minimum value and divided by the maximum value. While this fixed the contrast issue in a significant amount of data, it was very vulnerable to outliers which could take the maximum value. With this method, a single bright pixel in the image blocks the contrast normalization, and since small, very bright patches are not rare in our dataset, this method had to be improved. Instead, the maximum value was replaced by

the 98th percentile, a value for which 98% of the data is inferior. Using this value, the outliers are excluded from the normalization. The 2% of the pixels with values above the 98th percentile will be above one; thus, a ceiling was introduced where any value above one is replaced by one. With this normalization method, we ensure contrast equilibrium and a unitary range in the images. Figure 5.1 shows how the different normalization techniques affect the sections. Using the original image or the Min-Max normalization, such as Image 1 and Image 2, have relevant light contrast differences. However, when using the 98th percentile normalization, the difference in contrast between the two images is eliminated.

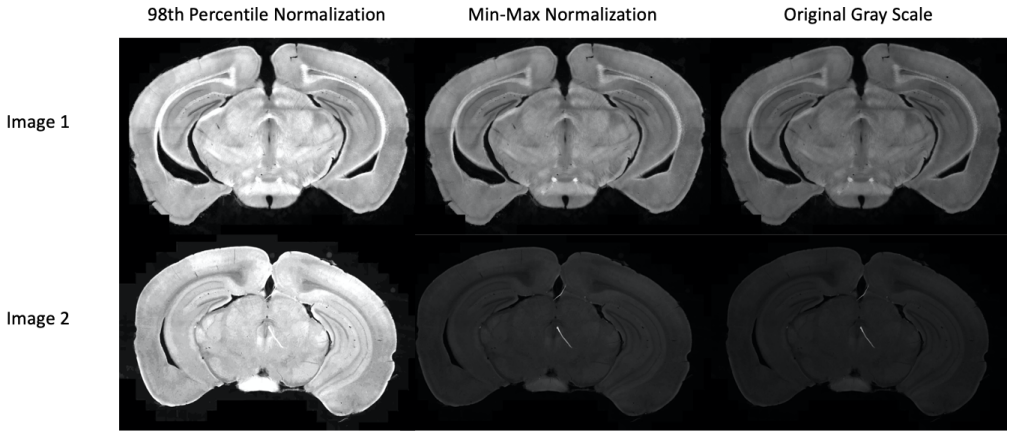


Figure 5.1: Effects of different contrast normalization functions

The training dataset will be transformed by data augmentation functions. These types of data augmentation were chosen by mimicking the artifacts or variations we found in our datasets. Due to a large amount of data, the images are loaded with the Keras' image generator functions which will load the images from memory in batches only when the batch is necessary. During training, each image is loaded once per epoch, and the data augmentation methods are applied then. The augmentation transformations have a random factor, and since they are applied to each image in every epoch, it is improbable that the exact same image is seen in training more than once.

Many of the common artifacts present in the section images end up partially occluding the image. One possible method to help overcome these issues is to force the model to be able to predict the correct label even with some of the features hidden and relying on the remaining visible characteristics to infer the section number. In the data augmentation step, we applied an algorithm that removes random patches from the image, simultaneously creating artificial artifacts and pushing the model to learn different features from each class. This algorithm was based on the work done by Zhong *et al.* [55], where randomly erasing image patches was used to increase a neural network's capacity to generalize. In our train dataset image processing, there is a 50% chance the artificial artifacts are applied. If an image is selected, a black patch with a rectangular shape is added on top of the image, blocking visibility to the pixels below. The patch's dimensions are randomly selected from a range of values with define both the area and the ratio. The area can range from 2% to 8% of the image's total area, and the patch's aspect ratio ranges from 0.3 to

2.5. The generated patch is inserted in the image at a random position. These ranges were selected experimentally while observing the effects of different range values and decided on the ranges that better approximated the real artifacts.

Since the slice images are not always vertically aligned, the network must ignore slight tilts of the image itself. A random rotation was then applied, with angles ranging from 0 to 10 degrees. With the single purpose of creating more diversity in the images to help to prevent the model from overfitting, the images are randomly shifted - both horizontally and vertically - with a maximum shift of 10%, and a horizontal flip is introduced with a 50% chance. We also applied these three image transformations to the data augmentation processes.

The data augmentation processing was only applied to the training dataset, while our custom normalization was applied to all the images, meaning that treating the data with our normalization function is necessary to use the model in production.

5.1.4 Model Architecture

Throughout the development of the regression models, different architectures and hyper-parameters were explored. Since no relevant differences were observed in prediction time, the criteria for architecture selection was mainly focusing on the size of the model and the quality of results. At this point, the implementation of the two CNNs began to differ. Deeper models seemed to benefit the quadrant regression but did not have the same impact on the single-label regression. After dropping architectures such as VGG16 [44], InceptionV3 [49] and Xception [14] due to unappealing results and large dimensions, the focus was shifted to the MobileNetV2 [42] and ResNet [23] architectures. The MobileNetV2's main benefit was the size. With only 3,538,984 parameters, resulting in a size of 14 MB, it showed positive results with the Single Label Classifier model. The same was not seen in the Quadrant Label Classifier. Likely due to the in-class diversity and more complex identifiable features, the regression only presented favorable performance with deeper architectures such as ResNet50V2 and ResNet101V2. To decide on the best architecture for each model, a systematic analysis was done throughout the project, which can be seen in the results chapter of this document.

The most significant improvement came from switching to pre-trained models, setting up the first layers as non-trainable and only training on the top convolutional layers. Here it was again stated that the quadrant regression needed a larger amount of training parameters when compared to the first regression model. The pre-training of the models was done on the ImageNet dataset [16]. The optimal range for training was between the 20% to 30% top layers, which included both convolutional and fully connected layers.

Tuning the learning rate hyper-parameter also proved to have a significant effect on the model's performance. To assist in finding the correct values, a method first introduced by Smith [45] and based on the learning rate finder implemented in the fast.ai library [28] for PyTorch [36] was used. These functions allow for plotting the effect of changing the learning rate on the loss values, making it visually comprehensive. Along with the discovered values, changing the learning rate

throughout the training can positively influence CNN's performance. Using Keras' [13] learning rate scheduler callback method, two different custom learning rate scheduler functions were applied and tested - cosine decay [33] and one cycle policy [45].

Independently from any factor explored during the project, there was always some visible oscillation in the model's performance during training. In order to take advantage of this oscillation, a model checkpoint callback was added to the train. The purpose of this callback is always to save a model in a particular epoch if the obtained result was the best so far. This helped to improve the model without having to guess the optimal number of train epochs. We used the validation's mean absolute error as the metric for saving the checkpoints.

The loss function used in the regression model was a mean squared error. We consider this to be the adequate loss function for our problem. Since the defining features for different section numbers are continuous, a prediction should be more penalized the further it was from the actual ground truth. We used a RMSprop optimizer and a batch size of 64.

5.1.4.1 Evaluation Metrics

For the model's metrics, a few different methods were used. The mean absolute error is used as a general metric of performance. By calculating the average error, it easily states how good the model is at executing the task. When using a regression model, the predicted values are not necessarily integers, meaning that in our case, the predicted section number can be a floating-point number in between two actual section numbers. Rounding the numbers to correspond to the traditional section numbers is an option; however, it did not show any benefit in either atlas generation - once we are generating our own atlas and it allows for in-between section numbers - or in evaluating the performance. Instead, to evaluate the accuracy, three different functions are also used to understand the performance, named Accuracy 1 (Acc1), Accuracy 2 (Acc2), and Accuracy 3 (Acc3). Accuracy 1 will assign positive values to predictions with an error below 1 when subtracting the predicted value to the ground truth. Accuracy 2 and 3 are similar, the difference being that they accept errors below 2 and below 3, respectively. To avoid any error inherent in certain classes, the average error of each section number is also plotted. This assists in detecting problems with the data, which, once fixed, can increase the CNN's quality. To evaluate the single label model, a group accuracy metric is also used. After converting the section numbers into one of the three groups - A, B, or C - it computes the accuracy of correct group attribution to each image. Since this is the main task of the model, this metric is of high importance.

5.2 Group C Segmentation

When using the pipeline, the conversion from full brain section to quadrants must be automatic. While Group A sections can be divided by the images' dimensions, dividing the Group C sections is far less trivial. The proportion of the cerebrum and brain stem regions is inconsistent and having missing regions highly influences the splitting process, which makes the quadrant splitting challenging.

An approach based on the image proportions was explored by defining ranges of aspect ratios where the different types of images would fall. The idea was that images only consisted of the brain stem would have an aspect ratio closer to 1 (squared), while images with one of the cerebrum regions would have a slightly more of a rectangular aspect ratio, and the images with all the regions would be a narrower rectangle. However, not only the accuracy of this method was poor, but even when it was correctly classified, the quadrant splitting was flawed as well. Due to the inconsistent proportions of the regions, there was no fixed line where the split should be done. Furthermore, when only one of the cerebrum regions was present, there was no method to distinguish which patch corresponds to the brain stem and to the cerebrum.

Since the dynamic programming methods did not reach usable results, the focus was shifted to building a segmentation dataset to train a neural network. We manually labeled 295 Group C images, creating segmentation masks with four classes: background, brain stem, left cerebrum, and right cerebrum.

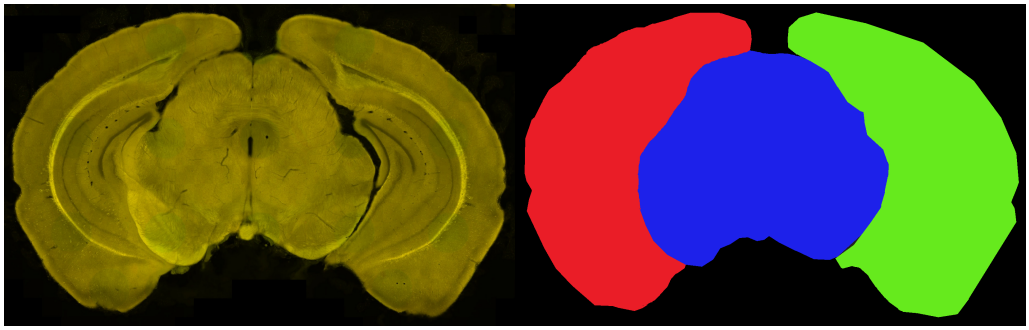


Figure 5.2: Group C section and corresponding ground truth mask

The training images were converted to grayscale and normalized with the 98th percentile normalization function developed for the regression CNNs and were resized to 256 by 256 pixels. Since the mask must match the original image, it was also resized to 256 by 256 pixels with a nearest neighbour interpolation. This type of interpolation is necessary to ensure all the pixels represent a class and no new values are accidentally generated while resizing. The model was built following a U-Net architecture [40], with a MobileNetV2 [42] as the base model for the encoder.

The segmentation model very accurately estimates the regions in an image. However, the generated masks often contain noise in the predicted regions. There should be at most one continuous area per region present, but the areas are generated with some pixels representing a different label inside a closed region area. The most likely motive for these errors is the small amount of data available for training. While more data would be possible to label, we found that the inserted noise could be easily removed with some image processing functions. Once the noise was removed, the masks had satisfying quality, and they perfectly fitted the task.

To remove the mask's noise, we first considered that there should only be one single area per region. Each mask is initially divided into three binary masks, one for each region category,

simplifying the image extraction. Then, each one of the binary masks passes through an outlier removal algorithm. To remove the outliers, the algorithm uses scikit-image's measuring functions [51] to count the number of individual areas. All the areas except for the largest one are deleted.

Since some regions are too small to be assigned a section number, a minimum size threshold filters the masks. Through experimentation, we decided that the minimum size for a region to be valid would be to constitute at least 10% of the image's area. If a region does not comply with this requirement, its quadrant is not extracted. Since the quadrants are going to be saved as images, the image should be divided into rectangular quadrants. Thus, the image is cut on the outer pixels of the image. In the case of the cerebrum, the regions are further divided into top and bottom quadrants. When both quadrants are present, we divide the image in half. However, we also consider that it may happen that only one of the quadrants - either top or bottom - is present. To deal with that, the lowest height coordinate and highest height coordinate of the mask are used to calculate the extent of the region. If the lowest point of the cerebrum region is not in the bottom 60% of the image, we consider that only the top quadrant exists. Symmetrically, if the highest point does not belong to the top 60% of the image, only the bottom quadrant is present. Figure 5.3 represents the mask processing method we apply.

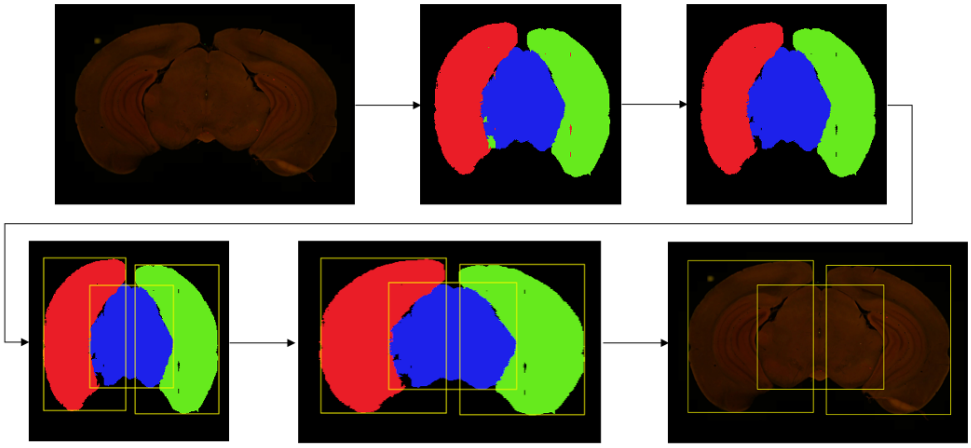


Figure 5.3: Segmenting process from predicting the mask, cleaning outliers and box generation. (1) Original image; (2) Predicted noisy mask that fits the general shape but with small incorrect patches; (3) Processed mask with at most one region from each category; (4) Boundary boxes for each region; (5) Re-scaled mask with region boundaries that fit the original image; (6) Region boundaries applied to the original section

Chapter 6

Atlas Generation

The focus of this chapter is to give some insight into the atlas generation process. To do so, we will present the options provided by the Allen Institute of Brain Science [18] and explain our methods that allow conversion from quadrant labeling values into a 2-dimensional reference atlas. With the atlas extraction algorithm, we can generate an atlas by inputting the slice number, α-angle and, β-angle or, in the case of Group C atlases, by inputting the left and right cerebrum numbers, the Q5 number and, the left and right β-angle.

6.1 Allen Mouse Brain Atlas

The Allen Institute contains multiple reference atlases, which vary in both contents and format. An atlas is a diagram of a brain slice which presents the regions in that part of the brain. For our work, we opted to use the 3D atlas in NRRD format - Nearly Raw Raster Data - which allowed us to both manipulate the cutting plane and region depth. From the different versions of the brain model, the most recent adult mouse brain was selected. This annotation released in October 2017 was named CCF_V3. The highest resolution 3D model allows for the generation of images with 1140 by 800 pixels and with 10 subsections between section numbers. This means that between consecutive section numbers, we can extract 10 different intermediary reference atlas. For example, between slice number 40 and 41, we can extract the atlases 40.1 to 40.9 which interpolate the two section numbers. The model can be seen as 1320 by 1140 by 800 volume where each cell has only one number, specifying the region ID. To be able to visualize a 2D atlas, we must extract a slice from the 3D model and replace the IDs with unique RGB values. Thus, it is necessary to create a color code that attributes a color to each region.

The regions in the mouse brain are hierarchical. Each region has a parent, and it is a sub-region of its parent. Due to the level of detail in the Allen atlases, representing each region individually - assigning a unique color for each ID - can result in atlases that are very difficult to use. Furthermore, for the purposes of our project, it is not necessary to observe the results in regions with very

high deepness. The Allen API developed a structure tree that allows for operations related to hierarchical relationships between regions. We concluded that using approximately a level 5 deepness for our analysis was the optimal solution. The level of regions is defined by their position in the tree, where we consider level N to be a region with N ancestors.

To use Allen's structure tree, we start by defining an instance of the ReferenceSpaceCache class, available in Python's Allen SDK. Using the get_structure_tree method of the reference space object, a hierarchical tree is provided along with a set of useful search methods. For the purposes of this project, the get_structures_by_id and parents methods are the most used. The first method allowed for the conversion between the unique identifier into an object containing the region's vital information while the latter returns the same information about a region's parent.


```
{'acronym': 'MOB',
  'graph_id': 1,
  'graph_order': 380,
  'id': 507,
  'name': 'Main olfactory bulb',
  'structure_id_path': [997, 8, 567, 688, 695, 698, 507],
  'structure_set_ids': [112905828,
                        691663206,
                        688152358,
                        687527945,
                        12,
                        184527634,
                        167587189,
                        114512891],
  'rgb_triplet': [154, 210, 189]}
```


Figure 6.1: Object structure for Main Olfactory Bulb (MOB) region obtained with the Allen's API [18]

Figure 6.1 shows the objects provided by the Allen API's structure tree and contains all the information necessary to build a custom color code to specify the depth of the regions. The Main Olfactory Bulb is considered to be a level 5 region. This can be seen when looking into the structure_id_path attribute, where the region's path is composed of seven ancestors, each one with a unique ID. We can exclude the first item of the path since it refers to the root of the tree, which results in five regions plus the actual region itself, thus making the MOB region level 5. The object also contains an RGB triplet value as well. These color values are Allen's default color codes which are seen in the available atlases on their website. It corresponds to approximately a level 4 color code since most regions with more than four ancestors will have no color distinguish from their siblings. Figure 6.2 shows an atlas with the default color code, where the Somatosensory area - level 4 region - is represented with one single color, and all the sub-regions are not distinguishable.

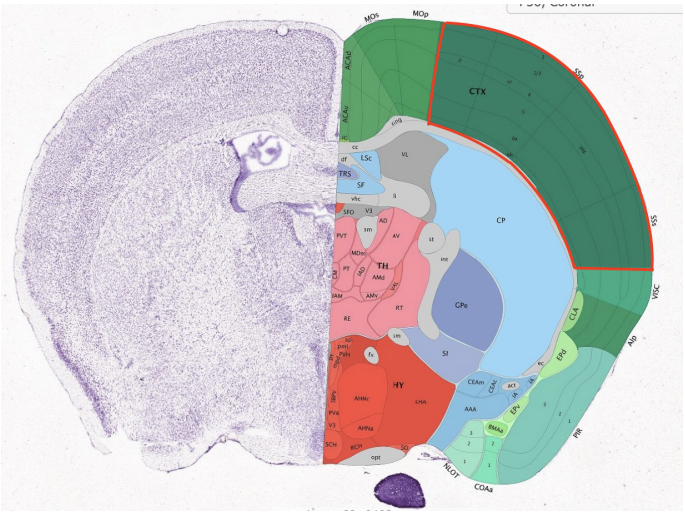


Figure 6.2: Somatosensory areas (SS) in highlight, without distinction between child regions. Image credit: Allen Institute of Brain Science [18]

6.2 Color Code Generation and Depth Specification

In order to generate the ideal atlases - that fit and adapt to the necessities of the clinical experts - it is necessary to create a custom color code. In later steps not included in this thesis, the color of a region will be its ID. Upon performing registration, the neuron distribution will be transferred to the atlas, and the region containing the neuron will be identified by color; thus, each region must have a unique color. Similarly to Allen's default color code, it is necessary to limit the number of sub-regions, meaning that at some point, the regions should stop being sub-divided. Figure 6.3 shows how using different depth specification changes the detail of the regions. Atlases with high depth (right), contain a lot of information about the sub-regions while less deep atlases (left) offer a simpler overview. Depending on the goal of the project, the importance of the sub-regions may vary, and thus, it is necessary to be able to control this aspect. Both images were extracted with our algorithm by only changing one parameter - visible region names.

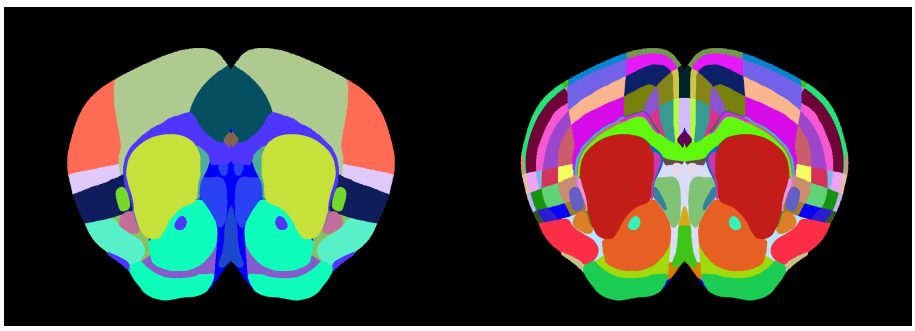


Figure 6.3: Effects of depth estimation in region detail

Initially, a version was implemented where the regions would be considered up to five ancestors. However, we found that level 5 did not always reflect the desired regions. To fix this issue, we decided that the user would provide a region list that specifies all the regions that should be seen. The clinical experts can decide on the set of regions that should have unique colors. We consider a region to be visible when it was specified and assigned a unique color. Any region that was not specified it is seen as part of the closest visible ancestor and it is not distinguishable from its siblings. In Figure 6.4 a branch of the tree is represented where the Isocortex is a parent with three direct children. The ORB, IL, PL, and Isocortex regions are visible while the remaining are not, meaning that regions as ORBvl, ORBvl1, or any other regions at this level will inherit the parent's color - ORB, in this case.

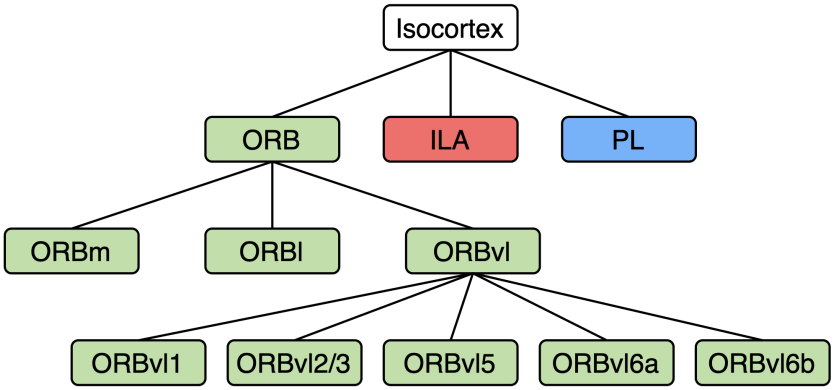


Figure 6.4: Region structure with custom color code considering specified visible regions

The brain model used for atlas generation was developed with the highest level of region detail (see Figure 6.3-right for highest region level coronal atlas). Our implementation of color code generation begins by creating a relation map between region IDs. The keys of the map are all the existing regions, and all the values are in the visible regions. The visible regions are assigned to themselves, while the youngest visible ancestor is assigned to the non-visible regions. Considering once more Figure 6.4, the corresponding value in the relation map for ORB's ID is ORB's ID, while the corresponding value for ORBvl1's ID is ORB's ID as well since ORBvl1 is a non-visible region. This map allowed us to create the level correlation before performing the ID to color conversion. In this latter step, instead of assigning colors to the IDs directly, the colors are assigned to the corresponding value of the ID in the relation map.

When generating a new color code, two random colors are assigned for each one of the visible regions. The generator assures that no colors are repeated. Two colors are generated for each region instead of one because our system allows for atlas generation with left-right differentiation, meaning that the left and right halves of the atlases have different color codes with no overlapping colors. Then, we assign a color to all regions, visible or non-visible, by merging the relation map with the visible region map. To assign a color to a non-visible region, the non-visible region's ID

is converted to its corresponding visible region's ID in the relation map. This corresponding value is converted into a color since all the visible regions are present in the color map, and the color is assigned to the non-visible region, assuring that all regions have a color assigned and the level conversion takes place. Figure 6.5 shows an example of the color map creation. The IDs 45, 63 and, 78 in this case represent brain regions present in the atlases. Firstly, in the relation map, we have the region IDs as the keys - 45, 63 and, 78. When the region is specified as a visible region - 45 and 78 - its corresponding value is itself. Otherwise, the value will be the youngest visible ancestor, as it happens in regions with ID 63. The color map then attributes a color to each visible region. To create color map for all regions, the two tables are merged, assigning to each region the correct color. The regions with IDs 45 and 63 have the same color assigned since region 63 is not visible and region 45 is the youngest visible ancestor.

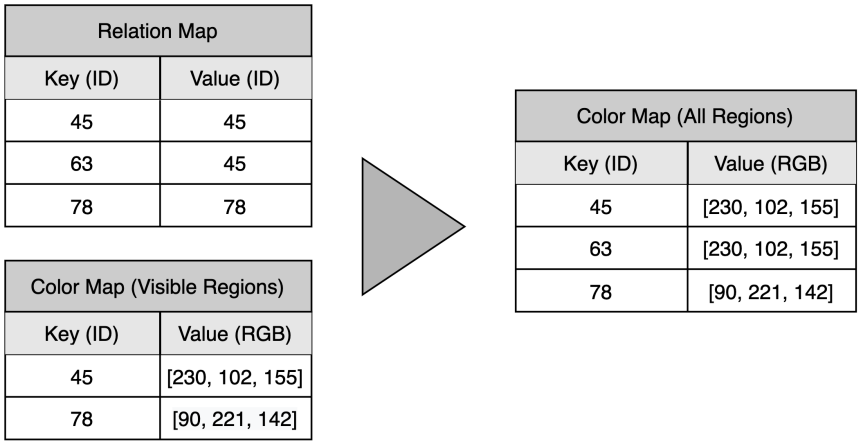


Figure 6.5: Merging relation map with visible regions color map, resulting in a color map with all the regions

The color map is then saved into JSON files. Upon starting a new atlas generation task, both creating a new color code or importing it from a JSON format previously saved are possibilities.

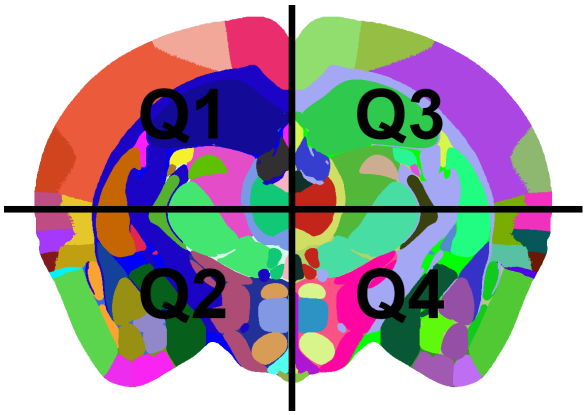
6.3 Quadrant Labeling to Atlas Values

As a result of our Quadrant Regression CNN, a set of quadrant values is obtained for each brain section. While quadrant section numbers and cutting plane are heavily connected, the conversion from one to another is not direct. Furthermore, this conversion must account for errors in the predictions originated in the models.

To simplify the relationship between quadrant values and atlas angles, we decided to measure the atlas tilts in section number difference. This means that to extract the angle α, it is only necessary to discover the difference between section numbers of the left and right halves. Similarly, the β-angle is obtained only by subtracting the bottom half section number to the top half section number.

Let us first only consider Group A sections, where four quadrants are predicted. We consider the section number of each quadrant to be the distance from the origin to the geometric center of the quadrant image, the z-coordinate. For a complete section image - without missing quadrants due to artifacts - the CNN returns what is equivalent to four points with three coordinates each, x, y and, z, where x and y are obtained geometrically by calculating the quadrant image center and z is predicted by the model. To reveal a plane, it is only necessary to have the coordinates of three points. Ideally, all our four points would be aligned in the same plane; however, this cannot be taken for granted due to the common appearance of small deviations or outliers. To account for this issue, our approach uses a sequence of techniques to average out the small deviations and exclude significant errors.

Using the four labels, it is possible to calculate two instances of both the α-angle and the β-angle. As Figure 6.6 suggests, we consider αA to be the top half left-right angles and αB to be the bottom half left-right angles. The same process generates βA and βB as well. When no errors occur, the A and B instances of the angles have approximate values; thus, we consider the final angle to be the average between the two versions.



αA	Q1-Q3
αB	Q2-Q4
βA	Q1-Q2
βB	Q3-Q4

Figure 6.6: Quadrants and angles in a section atlas

To obtain a usable conversion system, we must account for the possible errors or inconsistencies in the quadrant predictions. Since each quadrant label is predicted individually, plane consistency is not guaranteed. The first consideration to resolve this issue was to remove one of the quadrant labels from the plane generation. A plane can be defined by three points and so, by removing the least correct label, the plane could be settled. However, it was not possible to find an adequate method for identifying the quadrant to discriminate and another approach was taken.

Still only considering Group A sections, our conversion method uses averaging techniques and thresholds definition. The first step is to check for the present quadrants. If one of the quadrants is not present, the remaining three will be used to infer the α-angle and β-angle. Two quadrant may also be missing from the image. When this happens, the left or right side of the slice was lost, and so we can consider the α-angle to be zero and the β-angle is calculated by subtracting the two present quadrants. In the case of the four present quadrants, the conversion becomes more

challenging. Since both the α-angle and the β-angle are calculated through similar process, let us only consider the α-angle. We start by calculating both alphaA and alphaB and comparing them. If the absolute difference between both angles is below a certain threshold, the angles are consistent and the α-angle is assigned the average between the two values. When the difference between αA and αB is larger than a threshold of 1, the angles are not consistent and the incorrect angle must be removed. At this point, the predicted value in the Single Label Classifier can be used to untie the two angles. We extract the average of the two labels that compose each of the α-angles. The two average values are compared to the predicted single label value and the α-angle whose average is most similar is selected as the final α-angle, removing the outliers. Figure 6.7 represents the conversion method.

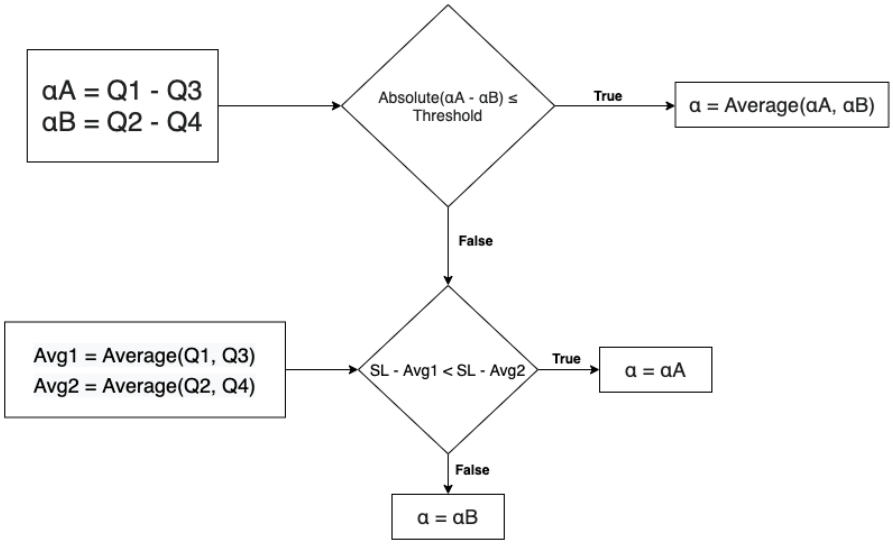


Figure 6.7: Converting quadrant values to angles. Q1, Q2, Q3 and, Q4 correspond to the quadrant values and SL to the section number predicted in the Single Label Classifier

6.4 Plane Extraction

6.4.1 Single Plane Extraction

Once the atlas position, α-angle, and β-angle have been extracted, the values are passed to the atlas extraction algorithm. As previously mentioned, the 3D model has 1140 voxels in width, 800 voxels in height and, 1320 voxels in depth. If a slice is extracted from the volume with a constant z-coordinate, the obtained image corresponds to a perfect coronal slice. This image representation is a 2D grid of voxels where the operation can be performed on its rows and columns. The 1320 voxel depth reflects the 132 coronal section numbers with 10 intermediary slices between two consecutive section numbers. We found that the corresponding z-coordinate is best represented by multiplying the section number by 10 and subtracting 7, e.g., the atlas number 67 is obtained by extracting all the voxels with the z-coordinate equal to 663 (67 x 10 - 7). Since the angles are

measured in the difference between slice numbers, the angle values are also multiplied by 10 to compensate for the intermediate slices.

When a section has no tilt, the correct atlas can be extracted by defining one single z-coordinate. However, when dealing with both α-angles and β-angles, it is necessary to vary the atlas' z-coordinate along with the columns or rows, with the objective of obtaining an x-y grid with values coming from different z-coordinates. The 2D atlas extraction process begins by calculating the angle steps, which define the amount that should be added to the z-coordinate in each row or column. Since the angles are passed in half section numbers difference, i.e., how many slices one half is ahead or behind the other, it is possible to obtain the increasing or decreasing steps that set the pace for tilting throughout the rows (α-angle) or columns (β-angle). Figure 6.8 represents a row of an angled atlas. The half section number is relative to the center of the quadrant, which means that the angles represent the variation range in only a portion of the atlas, corresponding to approximately half of the full length. To apply the angle variation throughout the row, a step is obtained by dividing the angle value by the corresponding range, which is from the left midpoint to the right midpoint when considering a row. This step will then be added to the average position, fetching voxels from different z-planes. The initial position is obtained by subtracting the angle from the average position. In combination with the added step, this ensures that there is an even distribution on the plane while maintaining the correct position. The same process is simultaneously applied in the columns, using the β-angle and the atlas' height as reference values.

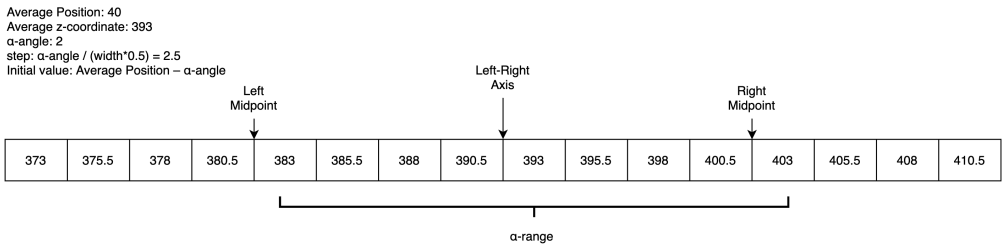


Figure 6.8: Representation of z-coordinate variations in row belonging to a atlas with an α-angle tilt

To deal with the right-left differentiation, the previously described process was divided into two parts. The algorithm first iterates through the left half of the volume, saving the left-half grid, and it then proceeds to generate the right-half grid. Since these values only represent the region ID, they must be converted into the correct RGB colors. By inserting each value into the color map previously created, we obtain the RGB image. The left and right atlas images are then merged, resulting in a dual-color code atlas.

As an extra feature and to assist in later analysis of the sections, our algorithm also provides an option to return the names of all visible regions present on the atlas. This is done by passing all the IDs present in the grid through the relation map, providing us with a list of IDs of the equivalent visible regions. From those values, the region acronyms are returned. Figure 6.9 shows

an example of a generated atlas with left-right differentiation.

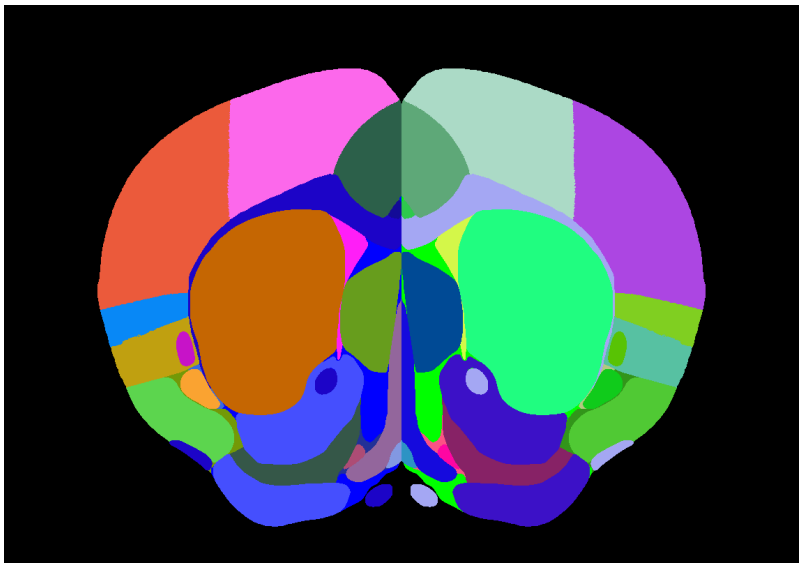


Figure 6.9: Single plane atlas with left-right differentiation

6.4.2 Multi-Plane Extraction

Due to the tissue separation occurring in the Group C sections, an angled atlas may not be enough to represent a section image. An algorithm heavily based on the single plane extraction method was developed to allow the extraction of one off-plane region. The region in question, the brain stem, is positioned in the center of the atlas. The multi-plane extraction further allows for the extraction of the brain stem region in any other z-plane. This was accomplished by defining all the regions (or sub-regions) either descendants of the brain stem or only occurring in those regions. These defined regions are eliminated from the cerebrum focused atlases. Since the cerebrum regions are narrow, the β-angle does not present relevant information. Thus, only the α-angle is considered. We also allow the possibility of extracting atlas with different α-angles in the right and left cerebrum, since the region movements might demand such requirements. After the regional atlas are merged, we obtain a multi-plane atlas that can be used in the Group C sections. Figure 6.10 shows an example of a generated atlas in the multi-plane sections.

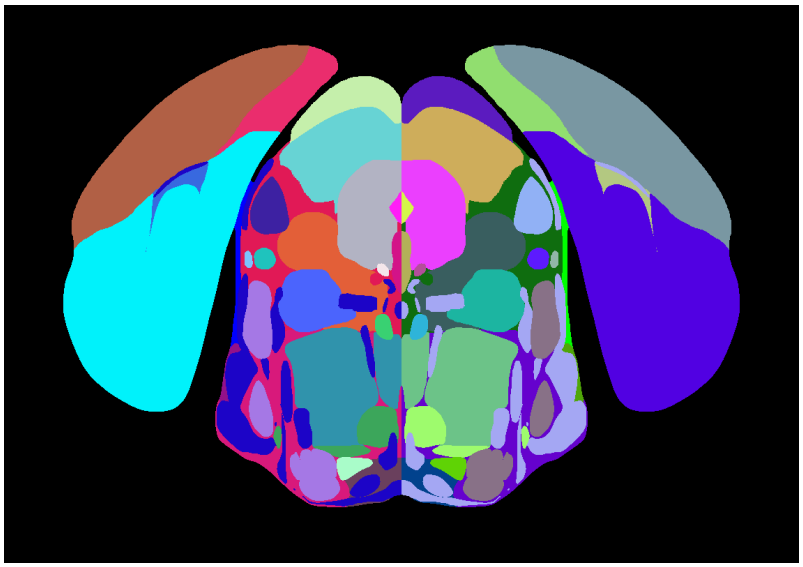


Figure 6.10: Multi plane atlas with left-right differentiation

Chapter 7

Results and Discussion

The development of this project led to a functional automatic system for section detection. Besides the achieved results, systematic comparisons between the tested models were progressively made. During this chapter, we will discuss the final results and the progress that built up to them.

7.1 Single Label Classifier Results

The single label classifier's results were remarkable. As a consequence of the reliable performance, the model ended up with a bigger role than initially expected and its predictions were used as the position value in the atlas generation a step. Since the improvements were made gradually, we will present the improvements applied at each development stage.

Analyzing the data is a fundamental part of developing a deep learning model. Observing the dataset and the location of the model's errors allowed us to identify some regular mistakes in the CNN's prediction. Through this method, we discovered the importance of normalizing the contrast in our data since the model was more likely to fail on darker images. While the error was reduced when proper normalization is applied, darker images are still more likely to confuse the model. This is due to the lower quality of the images since the low light image capturing results in a lower level of detail which can not be fixed by contrast normalization. We also observed that some isolated images with section numbers between 25 and 35 were assigned section numbers below 20. This was caused by an image artifact that can occur in any section number up until 35. While missing the left or right side of the section is very common in the Group B sections, the same vanishing hemisphere occasionally occurred between 23 and 35. The model predicted these half slices as Group B sections since this artifact is more common in that area. To fix this issue, some sections were selected and cut vertically. These half images were added to the dataset, which made the missing half artifact common in lower Group A sections. The model then ceased to focus on the artifact and assigned the labels based on the relevant features, eliminating this issue.

The first significant improvement regarding model architectures was observed when the lower layers of the models were pre-trained and set to untrainable. Using the ImageNet pre-training and fine-tuning on the top 20 to 30% layers, a clear shift occurred in the model's performance. Table

7.1 shows a comparison between two train runs on a ResNet50 architecture [23] with and without fine-tuning.

The metrics used in the tables below were defined in Section 5.1.4.1.

Model Training	MAE ↓	Group Acc (%) ↑	Acc1 (%) ↑	Acc2 ↑ (%)	Acc3 (%) ↑
All layers	2.84	90.32	41.94	63.44	75.27
Top 20%	1.79	91.40	40.86	68.81	80.64

Table 7.1: Pre-trained model's performance on a test dataset when training all the layers against train only the top layers

Using a fine-tuning method also released the model from larger architectures. When training the models from scratch, the best results were obtained using architectures with many parameters such as the ResNet152V2 [24]. Once the fine-tune methods were applied, the larger networks did not show any improvement.

The subsequent focus was on settling for a model architecture. Since the pre-trained models were being used, it was convenient to apply our model to the architectures present in the Keras API [13]. Being this model the first in the pipeline with the initial exclusive goal of dividing the sections into the corresponding groups and considering that two more models also take part in the pipeline, the size of the CNN was a relevant factor. Thus, three architectures of acceptable size were compared. Considering that the main focus was on the group accuracy, both the ResNetV2 model and the MobileNetV2 model performed at the same level. Thus, as the latter had a significantly smaller size and number of parameters, the development of the single-label classifier followed the MobileNetV2 architecture.

Architecture	MAE ↓	Group Acc (%) ↑	Acc1 (%) ↑	Acc2 (%) ↑	Acc3 (%) ↑
ResNet50V2 [24]	1.78	94.62	30.12	70.96	88.17
Xception [14]	1.80	89.25	43.01	70.97	86.02
MobileNetV2 [42]	1.68	94.62	29.03	66.67	83.87

Table 7.2: Performance of different architectures on a test dataset

The available data increased throughout the project. Whenever it occurred, the model's performance visibly improved. The introduction of Dataset 3 into our training dataset was possibly the most relevant step to reach the final model with suitable performance. Nonetheless, tuning the learning rate hyper-parameter proved to be crucial in both performance and training stability. A more systemic comparison is presented in Section 7.2. The learning rate finder method had a high impact on understanding the influence of the learning rate in training. While initially the linear scaling learning rate proposed by He *et al.* [25] was being used to find a default value for the hyper-parameter, it was later observed that while the formula may be suitable for some classification models, it was not for our regression model with a mean squared error loss function. Linear scaling learning rate draws an interesting insight into the relationship between learning rate and the batch size; however, the proposed formula did not account for the dimension of the loss values. Using mean squared error, the losses obtained are much higher than most commonly used loss

functions, such as categorical cross-entropy. Considering that the learning rate value is directly applied to the loss, we found that the learning rate should be lower with larger loss functions. Plotting the effects of changing learning rate on the loss, the graphic presented in Fig. 7.1 was obtained. The learning rate values should vary between the values that limit the negative gradient.

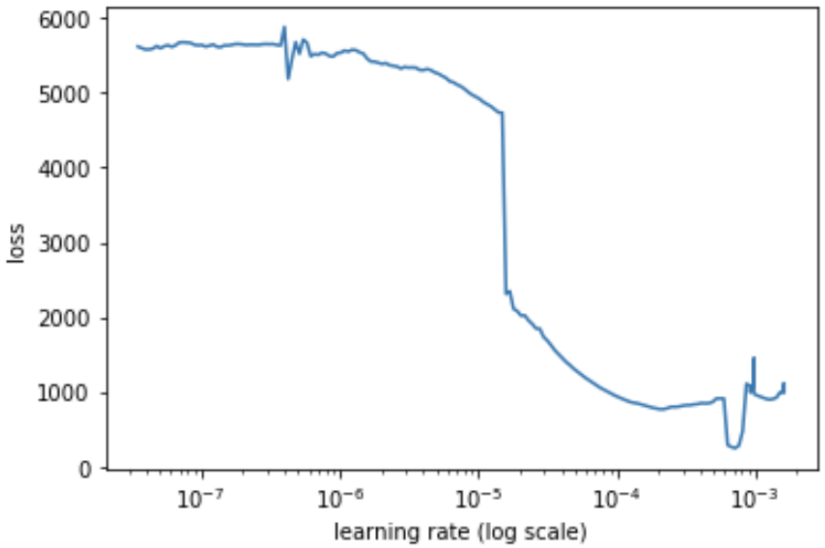


Figure 7.1: Learning rate finder of a MobileNetV2 model applied to single label regression

This model's training learning rate ranged between $2 \times 1 \times 10^{-5}$ and $2 \times 1 \times 10^{-4}$ with a one-cycle policy scheduler. Figure 7.2 shows the learning rate value along the training epochs. The learning rate linearly increases from the minimum learning rate to the maximum learning rate. Once it reaches the maximum values, it symmetrically decreases until it reaches the minimum. Finally, it again linearly decreases from the minimum learning rate value to zero.

In Table 7.3 the performance of the single label regression CNN in the different datasets is presented.

Dataset	MAE ↓	Group Acc (%) ↑	Acc1 (%) ↑	Acc2 (%) ↑	Acc3 (%) ↑
Train	0.63	98.09	79.89	96.80	99.70
Validation	0.97	97.96	67.96	90.54	97.20
Test	0.71	95.70	76.34	94.62	97.85

Table 7.3: Results of single label classifier

By analyzing the values of the group accuracy, we can conclude that it performs group classification with high accuracy. While the MAE values were also satisfactory, they did not give us a lot of insight into how the errors are occurring. However, by comparing the three accuracy metrics (Acc1, Acc2, and, Acc3), we can conclude that approximating the section number to a difference of 2 or 3 is a task that the model accomplishes consistently. The main difficulty lies in narrowing

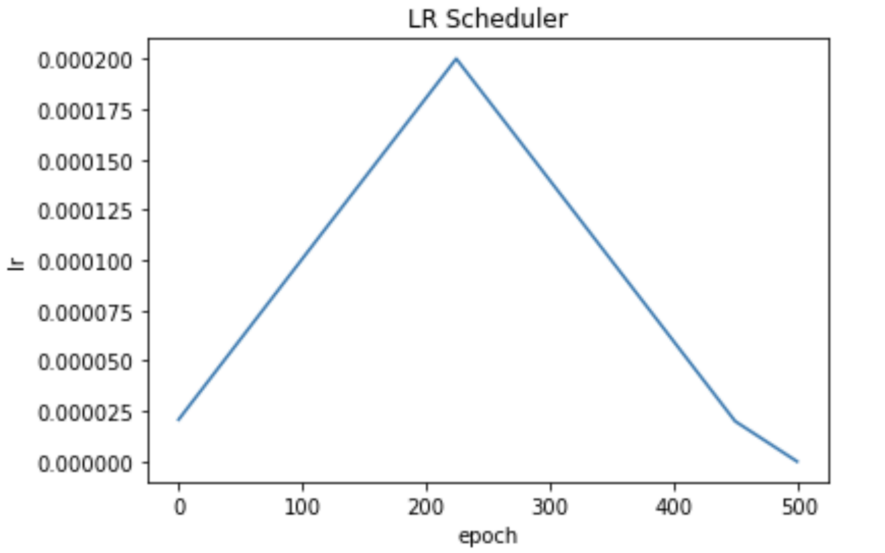


Figure 7.2: Learning rate scheduler for single label regression

the predicted section number to the exact value. Since human classification is also not entirely accurate, making the training data partially ambiguous, predicting the exact section number is quite challenging. Thus, considering the difficulty of the task and the results obtained in the different evaluation metrics, we can consider the model reliable at both single label section detection and group classification.

7.2 Quadrant Label Classifier Results

The quadrant regression model proved to be more challenging. The in-class diversity and lack of context had an impact on the model's performance. Nonetheless, it was possible to develop a model with appropriate results.

Because it was developed after the single label regression, some of the knowledge learned was directly applied. In particular, retraining all the layers of the CNN was quickly discarded after some experimentation. It was evident that, just like the first model, training the top layers of an ImageNet [16] pre-trained model was highly valuable.

Contrary to the full section model, denser CNN architectures seem to improve the model's quality. Table 7.4 shows the results of two different architectures' performance on a test dataset that consists of two full mouse brains separated from DS2 and DS3. The CNN showed a relevant improvement when changing to a larger model, nearly tripling the number of non-trainable parameters and increasing the number of trainable parameters by a third. Other model architectures, such as the MobileNetV2 [42] previously used, did not result in usable results, and thus, it was discarded.

The learning rate finder was again applied to the quadrant label regression. Table 7.5 shows the difference between two different learning rate values and schedulers. LR1 was based on He *et*

Architecture	MAE ↓	Acc1 (%) ↑	Acc2 (%) ↑	Acc3 (%) ↑	Trainable Parameters	Non-Trainable Parameters
ResNet50V2	1.84	33.65	65.39	82.14	15,234,048	8,330,752
ResNet101V2	1.64	36.52	70.33	87.56	19,179,008	23,447,552

Table 7.4: Comparison of performance between two differently sized architectures

al. [25] formula for a starting learning rate with Loshchilov *et al.* [33] method of cosine decay while LR2 used the methods proposed by Smith [45]. Considering that no other variables were changed, the impact of the learning rate hyper-parameter can be seen. In our model, the LR2 method suited the task much better. As argued previously, the use of a mean square error for a loss function changes the contribution of the learning rate and, by using the methods proposed by Smith [45], the learning rate can be customized to a particular model, which proved to be a constructive feature in our code.

Run	MAE ↓	Acc1 (%) ↑	Acc2 (%) ↑	Acc3 (%) ↑
LR1	1.64	38.76	68.58	86.76
LR2	1.42	41.31	77.03	89.63

Table 7.5: Comparison of performance between two different learning rate setup in the same model, where LR1 corresponds to a cosine decay starting with a default value of 0.025 and LR2 corresponds to one cycle policy with minimum and maximum values of $2 \times 1 \times 10^{-6}$ and $2 \times 1 \times 10^{-5}$

Table 7.6 shows the final results of the regression on the different datasets. At this point, the effects of the random eraser data augmentation (see section 5.1.3) were compared. In Table 7.7, we can see the results of a similar model with the single difference of not using the random eraser data augmentation. Even though the differences appear to be minimal, we opted to use the model trained with the random eraser in our pipeline since it performed better in our large validation dataset.

Dataset	MAE ↓	Acc1 (%) ↑	Acc2 (%) ↑	Acc3 (%) ↑
Train	0.78	70.45	95.02	99.47
Validation	0.96	63.50	90.59	97.52
Test	1.42	42.74	74.96	90.75

Table 7.6: Final results of quadrant label regression

When comparing the Quadrant Label Classifier to the Single Label Classifier's performance, a similar conclusion can be taken relative to the model's reliability. In this task, the images are more vulnerable to data deformations, making quadrant classification not viable in certain images containing large artifacts. Since the test dataset has a high rate of deformed images compared to the train and validation datasets, the model's performance justifiably drops. However, the validation set, which was not used for training, is larger and balanced, allowing us the take conclusions on the

Dataset	MAE ↓	Acc1 (%) ↑	Acc2 (%) ↑	Acc3 (%) ↑
Train	0.79	69.16	95.54	99.53
Validation	1.00	59.72	90.16	97.37
Test	1.38	44.17	76.55	91.06

Table 7.7: Results of quadrant label regression without random eraser

model's performance. The model's results regarding Acc2 and Acc3 were again very accurate. The model's performance was satisfactory and we considered it stable for quadrant section detection.

7.3 Group C Segmentation Results

Evaluating segmentation models can be a challenging task. While visually interpreting the results is an important step, it can be too unreliable to guarantee the quality of the model. The model's final goal is to: (1) identify the present regions and (2) provide cropping coordinates that perfectly extract the regions.

All the metrics were applied after processing the predicted masks, cleaning the prediction errors (see section 5.2). After post-processing, the results were exceptionally good, especially when considering the small dimension of the dataset. Relative to the prediction of the present regions, our model performed with 100% accuracy in our test dataset. When comparing the predicted mask with the hand-drawn mask, the present regions were always the same. Understanding the role of the model in the pipeline allowed us to infer a metric that measures the quality of the model as cropping software. To evaluate the cropping quality, we extracted the cropping coordinates of both the predicted regions and the label-mask regions and performed a mean absolute error between the two lists of coordinates. The resulting MAE was 0.83, meaning that, on average, the cropping coordinates in our 256 by 256 masks have a 0.3% error.

Even though the predicted masks often contain many errors, the region boundaries are very accurate. Since the focus of this step is to predict the boundaries, the image processing methods managed to clean the output and convert it to appropriate results. Without the post-processing steps, the model's predictions would mediocre, showing the importance of developing functions that complement the deep learning approaches.

7.4 Pipeline Evaluation

Evaluating the pipeline consisted of understanding how accurately the position and orientation of the atlas were predicted. The end values of the deep learning branch of the pipeline were the position, the α-angle, and the β-angle, which can be inferred from the quadrant labels predicted. Thus, the final evaluation could be statistically done by comparing the true position and orientation of an evaluation dataset to the predicted parameters. With the single label value being used as the position parameter, we obtained a mean absolute error of 0.89, meaning that the prediction error

of the position is 0.7%. Still using the mean absolute error as a metric, we obtained an error of 0.76 on α-angle prediction and 1.35 on β-angle prediction.

Since the Q5 quadrant is assumed to be in an independent plane, it is relevant to perform a separate evaluation on this patch. Since we do not consider any angle in the Q5 quadrant, the only relevant label is the positional value. Using the same evaluation dataset, we obtained a mean absolute error of 1.83. This error translates to a 1.4% error in the prediction of said quadrants.

In Figure 7.3 we can see the results of the pipeline. For each of the images, the atlas was generated fully automatically. The visual results demonstrate a clear connection between the real image and the generated atlas, even in sections with substantial artifacts.

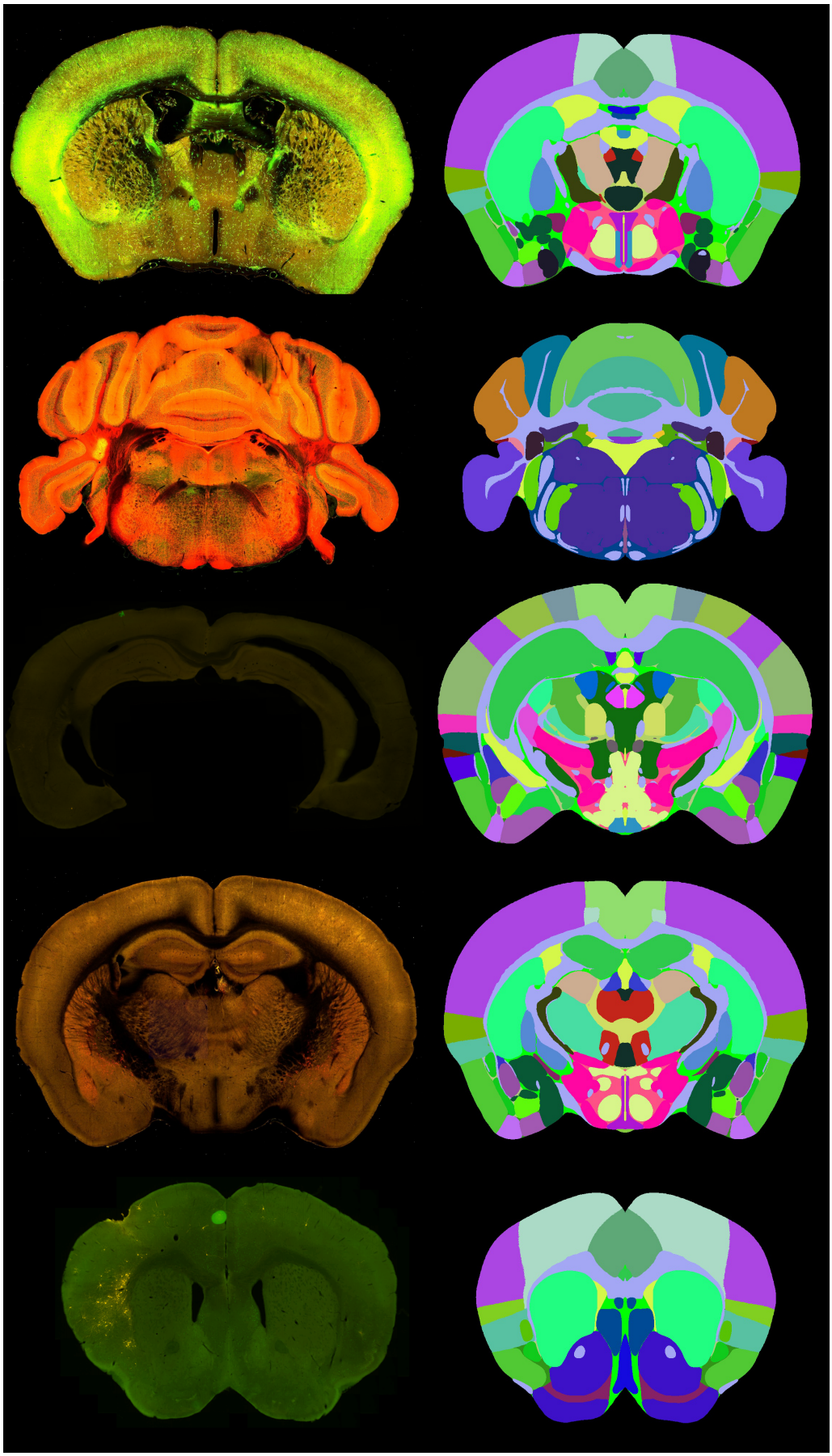


Figure 7.3: Results of atlas generation. The final tool takes experimentally-obtained section images (left) and automatically generates the appropriate atlas (right)

Chapter 8

Conclusion

8.1 Conclusion

Section detection is a challenging task both for human experts and deep learning models, and a certain degree of error has to be accepted in any of the cases. Nonetheless, minimizing the errors results in much more reliable analysis, which can significantly aid the system neuroscience projects it takes part in. Thus, there should always be a focus on error minimization when developing automatic section detection methods. Apart from the model's accuracy, the importance of an automatic or even semi-automatic section detection system must be highlighted. Manually classifying brain sections is a laborious and slow task, which can be astonishingly simplified when assigning or suggesting a number almost instantaneously. Even in the early stages of our work, the single label classifier was added to the software providing a suggestion to the human classifier. This addition allowed for faster manual classification since it immediately gave the classifier an estimation for the correct atlas number. When fully integrated, the pipeline produces results at a remarkable speed. A process that once took several hours for each brain can now categorize multiple mouse brains in a matter of seconds. Our average quadrant prediction time is 1.8 seconds per section, and taking less than 2 minutes to obtain the section numbers for all the slices from a mouse brain. Furthermore, our solution provides the feature of classifying and producing the angled atlases, which was previously unreachable due to the lack of an algorithm that allowed for a tilted atlas extraction algorithm and the increased labor attached to it.

The importance of data quality and quantity in deep learning projects is undeniable. It is evident that while some improvements can be made by modifying the specifications of the deep learning models, the most significant gains came from either increasing the dataset or cleaning up the data. This highlights the unsurprising statement that projects such as this one must have a strong focus on collecting, understanding, and fixing the datasets available. When in need of an increase in the data, data augmentation techniques assist in minimizing that necessity; however, it does not provide equal results to an increase of the dataset. Properly analyzing the dataset can also have a meaningful impact since some fixable issues can be identified, such as the difference in the images' contrasts or missing regions artifacts that suggested a wrong classification in our

data. Regarding data augmentation transformations, these should always remain in the context of the data. In our case, image shifts, rotations, flips, and randomly erased patched fitted into the diversity of our data since all of these deformations are present in our dataset.

Despite the importance of data, there are other changes that can be applied to leverage a deep learning model. Based on our results, we can highly recommend progressively applying the changes in the model one by one. Since CNNs take a significant amount of time to train, one might be tempted to apply multiple changes simultaneously. Only by stepping our development, we managed to locate the meaningful setups to our model while changing multiple factors from one train to another left us unsure about which of the applied aspects had an actual role in any change of the model's performance.

We found that transitioning from retraining the whole model to only training the top layers of a pre-trained model was the most significant improvement unrelated to the dataset. The architecture of the model also was of significant impact. There are multiple architectures to choose from, and it may not be trivial to select the correct ones for testing. Our architecture selection consisted of two parts; we first selected a small number of models that we deemed appropriate both in size and performance results in other tasks. Then, we systematically tested these different architectures. Although it is still time-consuming, reducing the number of possibilities makes the comparisons viable. The last major focus point was the learning rate adjustments made throughout the project. From our experience, this is the most important hyper-parameter. Investing some effort into understanding how to adjust the learning rate proved to be a simple and efficient way of improving the models. Even when the models correctly converge to the optimal values during training, there is a substantial chance that the best performance does not fall on the last epoch of training. To take advantage of this property, callback functions during training can be used to save the models that surpass the performance of previous epochs, allowing us to obtain the best model of the whole train run and controlling the unpredictable oscillations of the model.

Even with a small error, the inconsistencies of the quadrant label regression generated some challenges. In a common Group A section, four labels are predicted, which would ideally all belong to the same plane. However, since it is not possible to obtain a perfect model, errors must be accounted for in the conversion from predicted labels to the cutting plane. Accepting the existence of the error was a crucial part of our solution since it allowed us to focus on the development of an algorithm that can minimize or even eliminate the errors.

While interpreting the pipeline's results, on several occasions we found that the difference between the predicted values and the ground truth values was caused by mislabeling by the experts. In other cases, both the labeling and predicted values would be arguable. This uncertainty turned the evaluation task challenging, once it was not possible to assume the difference between the predicted values and the labeled values as the error. Our proposed solution to fix this problem in later work is to build an evaluation dataset that is labeled by multiple experts in order to eliminate biases and allow for the comparison of the tool's performance relative to human capability.

It could be concluded from the evaluation results that the tool was very consistent at predicting section numbers. This was observed when analyzing a series of consecutive sections from a

single mouse brain where we could see that the section numbers tended to increase - or decrease - correctly along the sections. While this leans to the existence of a small bias, it also positively shows a coherent prediction in the tool.

One of the best features of our classification system is its performance in dealing with deformed data. Resulting from our efforts to introduce both natural and artificial deformations into our dataset and the chosen steps that constitute the workflow, the pipeline is able to deal with difficult data. In particular, the Single Label Classifier performed very well even in very difficult sections. Due to the quadrant labeling method, missing tissue and highly disrupted quadrants have a small impact on the overall performance. At the same time, smaller artifacts that are inherent to this type of data do not seem to cause relevant disruption in the predictive performance. Comparing to image similarity methods that were used in previous research, our tool is more robust.

The segmentation step in the developed tool presents remarkable performance. Despite the size of the available dataset for Group C segmentation, it was possible the achieve our goal with practically no error. The post-processing methods have a great impact on the accuracy of this step, pointing to the importance of complementing deep learning models with image processing methods.

The architecture of our tool also allows for easy improvements upon the availability of new data, offering a progressively better accuracy without any structural changes. The tool's performance did accomplish our goal. Considering the difficulty of the task, introducing the automatic pipeline with this level of efficiency will drastically simplify the work done in systems neuroscience research.

The development of an atlas generation algorithm was a significant task during this thesis. Manipulating 3D brain models is a highly beneficial technique. The control to extract atlas at different angles, with custom region specifications and color codes is a noticeable improvement that can be used not only with our tool but also in other projects. With the added customization, the clinical experts can better fit the atlases to their needs, leading to better brain analysis.

The integration of our solution in the pipeline for mapping brain activation in the mouse undoubtedly improved the potential and efficiency of the software. The accurate and fast detection that our tool provides will allow for simpler and more efficient processing of mouse brain data. The achieved results were very satisfying, contributing to a large reduction of human workload in health research.

8.2 Future Work

The methods developed in this thesis can be generalized to fit structures other than mouse brains. An interesting development of our work would be applying the same pipeline to rat's or young mouse's brain structures, covering an even larger amount of neuroscience projects.

While we gained full control of the 3D reference atlas, fully fitting our requirements, the developed method relies on the existence of a 3D reference brain volume. Once 2D atlas sequences are more common than the 3D volumes, generating 3D atlas models from the two-dimensional

sequences may benefit future projects. This would allow for detailed atlas extraction from different modeling systems, releasing the project from the necessity of pre-made 3D atlases.

Since a significant insight is gained by evaluating the results, a better evaluation framework can lead to meaningful discoveries or improvements. To better evaluate the pipeline, it would be valuable to create a dataset that was labeled by multiple experts. This would not only reduce the individual bias but would also allow us to compare the tool's performance relative to the human capabilities.

8.3 Main Contributions

The results achieved with our work contributed significantly to automatic brain mapping. The end-to-end tool resulting from this project is integrated into the automatic pipeline for mapping brain activation in the mouse, developed by the Medizinische Universität Innsbruck [1], playing a crucial role in the software and getting in one step closer to accomplish fully automatic mouse brain mapping. Due to the promising results, our work will also be adapted and submitted for publication at the SPIE Medical Imaging Conference [3], contributing to future section detection and brain mapping research.

References

[1] Medizinische Universität Innsbruck - Medical University of Innsbruck, https://www.i-med.ac.at/msig/.

[2] Niedworok CJ, Brown AP, Jorge Cardoso M, Osten P, Ourselin S, Modat M, Margrie TW. aMAP is a validated pipeline for registration and segmentation of high-resolution mouse brain data. Nat Commun. 2016 Jul 7;7:11879. doi: 10.1038/ncomms11879. PMID: 27384127; PMCID: PMC4941048.

[3] SPIE Medical Imaging Conference https://spie.org/conferences-and-exhibitions/medical-imaging?SSO=1.

[4] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[5] Daniel Adler, John Pluta, Salmon Kadivar, Caryne Craige, James Gee, Brian Avants, and Paul Yushkevich. Histology-derived volumetric annotation of the human hippocampal subfields in postmortem mri. *NeuroImage*, 84, 09 2013.

[6] Abien Fred Agarap. Deep learning using rectified linear units (relu), 2019.

[7] Nitin Agarwal, Xiangmin Xu, and M. Gopi. Robust registration of mouse brain slices with severe histological artifacts. In *Proceedings of the Tenth Indian Conference on Computer Vision, Graphics and Image Processing*, ICVGIP '16, New York, NY, USA, 2016. Association for Computing Machinery.

[8] Bassam Almogadwy, Kenneth McLeod, and Albert Burger. A deep learning approach for slice to volume biomedical image integration. pages 62–68, 05 2019.

[9] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346–359, 2008. Similarity Matching in Computer Vision and Multimedia.

[10] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape context: A new descriptor for shape matching and object recognition. *NIPS*, 2, 07 2000.

[11] Facundo Bre, Juan Gimenez, and Víctor Fachinotti. Prediction of wind pressure coefficients on building surfaces using artificial neural networks. *Energy and Buildings*, 158, 11 2017.

[12] Zixuan Chen, Zekai Xu, Weiwei Yi, Xin Yang, Wenguang Hou, Mingyue Ding, and Oleg Granichin. Real-time and multimodal brain slice-to-volume registration using cnn. *Expert Systems with Applications*, 133:86 – 96, 2019.

[13] François Chollet et al. Keras. https://keras.io, 2015.

[14] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, 2017.

[15] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005.

[16] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[17] Enzo Ferrante and Nikos Paragios. Slice-to-volume medical image registration: A survey. *Medical Image Analysis*, 39:101–123, Jul 2017.

[18] Allen Institute for Brain Science. Allen mouse common coordinate framework. *Technical White Paper*, pages 1–18, 2015.

[19] Eli Gibson, José A. Gómez, Madeleine Moussa, Cathie Crukley, Glenn Bauman, Aaron Fenster, and Aaron D. Ward. 3D reconstruction of prostate histology based on quantified tissue cutting and deformation parameters. In Robert C. Molthen and John B. Weaver, editors, *Medical Imaging 2012: Biomedical Applications in Molecular, Structural, and Functional Imaging*, volume 8317, pages 206 – 212. International Society for Optics and Photonics, SPIE, 2012.

[20] Maged Goubran, Cathie Crukley, Sandrine De Ribaupierre, Terence Peters, and Ali Khan. Image registration of ex-vivo mri to sparsely sectioned histology of hippocampal and neocortical temporal lobe specimens. *NeuroImage*, 83, 07 2013.

[21] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour, 2018.

[22] Eldad Haber and Jan Modersitzki. Intensity gradient based registration and fusion of multimodal images. In Rasmus Larsen, Mads Nielsen, and Jon Sporring, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2006*, pages 726–733, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks, 2016.

[25] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks, 2018.

[26] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[27] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[28] Jeremy Howard et al. fastai. https://github.com/fastai/fastai, 2018.

[29] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[30] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, Tiegang Chen, Guangxiao Hu, Shaohuai Shi, and Xiaowen Chu. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes, 2018.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. Red Hook, NY, USA, 2012. Curran Associates Inc.

[32] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[33] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017.

[34] Piotr Majka and Daniel Wojcik. Possum-a framework for three-dimensional reconstruction of brain images from serial sections. *Neuroinformatics*, 14, 07 2016.

[35] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training, 2018.

[36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[37] H. Rivaz, S. J. Chen, and D. L. Collins. Automatic deformable mr-ultrasound registration for image-guided neurosurgery. *IEEE Transactions on Medical Imaging*, 34(2):366–380, 2015.

[38] Hassan Rivaz, Zahra Karimaghaloo, and D. Louis Collins. Self-similarity weighted mutual information: A new nonrigid image registration metric. *Medical Image Analysis*, 18(2):343 – 358, 2014.

[39] Hassan Rivaz, Zahra Karimaghaloo, Vladimir Fonov, and Louis Collins. Nonrigid registration of ultrasound and mri using contextual conditioned mutual information. *IEEE transactions on medical imaging*, 33:708–25, 03 2014.

[40] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, May 2015.

[41] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2015.

[42] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[43] Martin Simonovsky, Benjamín Gutiérrez-Becker, Diana Mateus, Nassir Navab, and Nikos Komodakis. A deep metric for multimodal registration. In Sebastien Ourselin, Leo Joskowicz, Mert R. Sabuncu, Gozde Unal, and William Wells, editors, *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2016*, pages 10–18, Cham, 2016. Springer International Publishing.

[44] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[45] Leslie N. Smith. Cyclical learning rates for training neural networks, 2017.

[46] Jun Ho Song, Woochul Choi, You-Hyang Song, Jae-Hyun Kim, Daun Jeong, Seung-Hee Lee, and Se-Bum Paik. Precise mapping of single neurons by calibrated 3d reconstruction of brain slices reveals topographic projection in mouse visual cortex. *Cell Reports*, 31(8):107682, 2020.

[47] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. 15(1), 2014.

[48] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[49] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.

[50] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):376–380, 1991.

[51] Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014.

[52] Jing Xiong, Jing Ren, Liqun Luo, and Mark Horowitz. Mapping histological slice sequences to the allen mouse brain atlas without 3d reconstruction. *Frontiers in Neuroinformatics*, 12:93, 2018.

[53] Zhengyi Yang, Kay Richards, Nyoman Kurniawan, Steven Petrou, and David Reutens. Mri-guided volume reconstruction of mouse brain from histological sections. *Journal of neuroscience methods*, 211:210–217, 09 2012.

[54] Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks, 2015.

[55] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation, 2017.