Development and Evaluation of 3D Autoencoders for Feature Extraction

Entwicklung und Evaluation von 3D Autoencodern zur Merkmalsextraktion
Bachelor-Thesis von Robin Hesse aus Frankfurt am Main
Tag der Einreichung:

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Prof. Dr. Moritz Helmstaedter
3. Gutachten: Dr. Marcel Beining und M.Sc. Dorothea Koert

TECHNISCHE UNIVERSITÄT DARMSTADT

Development and Evaluation of 3D Autoencoders for Feature Extraction
Entwicklung und Evaluation von 3D Autoencodern zur Merkmalsextraktion

Vorgelegte Bachelor-Thesis von Robin Hesse aus Frankfurt am Main

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Prof. Dr. Moritz Helmstaedter
3. Gutachten: Dr. Marcel Beining und M.Sc. Dorothea Koert

Tag der Einreichung:

In cooperation with Max-Planck-Institute for Brain Research

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 19. Dezember 2017

(Robin Hesse)

Abstract

This thesis addresses the problem of aligning connectome data. The advancing scale in connectome reconstruction introduces new problems. One particular problem are spatial shifts and distortions in the image data that make it hard to come to a global alignment. This problem can be solved by splitting the data into small sub-cubes and creating the connectome for every sub-cube isolated and subsequently merge the sub-connectomes to the global connectome. In this thesis the alignment of each sub-cube is achieved with an image registration approach that had an accuracy of 84% on real data. Moreover, a 3D shape descriptor for feature extraction is presented. It can be used to extract rich features for later matching algorithms. It combines the ability of a recently introduced neural network architecture called PointNet to work on point cloud data with an autoencoder and a cost function that guarantees invariance to permutations in the input. The unsupervised classification task on ModelNet40 achieved an accuracy of 83.1% and showed that the shape descriptor is able to extract useful features.

Zusammenfassung

Diese Thesis beschäftigt sich mit dem Problem des Alignements von Konnektomdaten. Die fortschreitenden Größen der dichten Rekonstruktion neuronaler Netzwerke bringt neue Herausforderungen mit sich. Eine spezielle Herausforderung sind Versätze und Verzerrungen in den Daten, die es schwer machen, die Daten global zu alignieren. Eine Lösung hierfür ist die Unterteilung der Daten in kleine Würfel in denen lokale Konnektome erstellt werden, die dann zu einem globalen Konnektom zusammengeführt werden. In dieser Thesis wird das Alignement jedes Würfels mit einem Bildregistrierungsansatz umgesetzt, der eine Genauigkeit von 84% auf echten Daten erreicht. Zusätzlich wird ein 3D Formdeskriptor zur Eigenschaftsextraktion präsentiert. Seine extrahierten Eigenschaften können für spätere Alignementalgorithmen benutzt werden. Er kombiniert die Fähigkeit einer neuen neuronalen Netwerkarchitektur auf Punktwolken zu arbeiten mit einem Autoencoder und einer Kostenfunktion, die Invarianz zu Permutationen in der Eingabe garantiert. Die unüberwacht Klassifikationsaufgabe auf ModelNet40 erreicht eine Genauigkeit von 83.1% und zeigt, dass der Formdeskriptor in der Lage ist, brauchbare Eigenschaften zu extrahieren.

Acknowledgments

Many thanks to Prof. Dr. Peters and Prof. Dr. Helmstaedter who gave me the opportunity to write this thesis.

Many thanks to my supervisors Dorothea Koert and Dr. Marcel Beining who always had an open ear and helped me so much.

And many thanks to all the inspiring people in the lab who made the last six months unforgetable.

Contents

Figures and Tables

List of Figures

Abbreviations, Symbols and Operators

List of Abbrevations

Notation	Description
i.e.	id est (engl.: that is to say)
e.g.	exempli gratia (engl.: for example)
3D	3 dimensional
EM	electron microscopy
SBFSEM	serial block-face scanning electron microscopy
SE	squared error
MSE	mean squared error
AE	autoencoder
DAE	denoising autoencoder
GD	gradient descent
px	pixel

List of Symbols

Notation	Description
$\boldsymbol{W}$	weight matrix
$\boldsymbol{b}$	bias vector
I/h/O	input/hidden/output layer of a neural network
$\boldsymbol{\mu}$	external transformation parameter
T	transformation
v	view
r	border of transformed view
σ	square root of variance

List of Operators

Notation	Description
ϕ	encoding function
φ	decoding function
$\mathscr{L}$	loss

1 Introduction

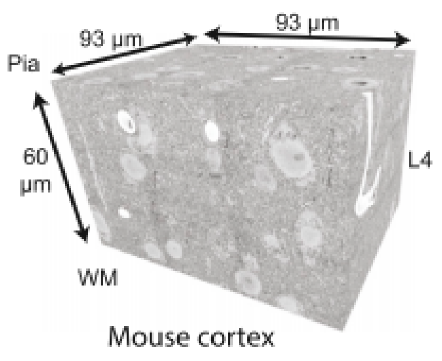
Although the human brain has been investigated for several thousands of years it remains one of the biggest mysteries in our body. This is most likely due to its sheer complexity. Each neuron has on the order of 1,000 synaptically coupled partner neurons (Helmstaedter, 2013), with whom it can directly and specifically communicate. This results in a total of approximately 100 trillion connections within the human brain and is the reason for its unique cognitive abilitys. Once we have understood all those connections and associated actions, we might be able to cure several brain disorders and simulate intelligence. The consequences are unimaginable. However, until then a long path is to go and, on this, we have to understand the very details of the brain. One step towards this goal is to completely understand the structural connectivity of the brain. A new research area has emerged that is exactly on that track: Connectomics.

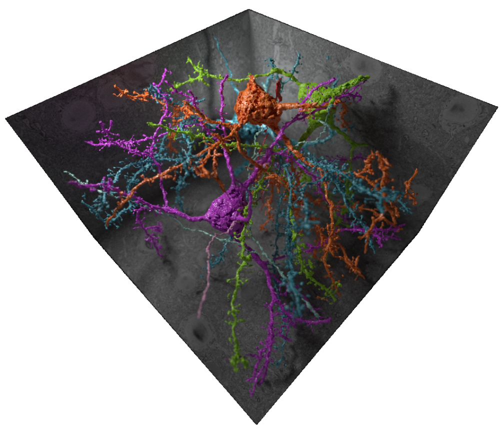
1.1 Connectomics

Connectomics refers to the study and production of connectomes. In the context of neuroscience, connectomes are communication maps of neuronal circuits. The underlying idea of connectomics is that the structure of a nervous system can expose important knowledge on its organization and even on functional properties. This argument is not obvious and often criticised. Nevertheless, there are good arguments to investigate further into connectomics (Morgan and Lichtman, 2013). Connectomes may be used for several objectives. They can provide valuable insights on similarities between structures in the cerebral cortex of different individuals from different species and help to gain information about algorithms of sensory perception. Moreover, they can be helpful to understand alterations in the brain in psychiatric diseases (Helmstaedter, brain.mpg.de). This could be an important step towards potential cures.
However, before gaining new information, the connectome of a brain first needs to be produced. Due to the density, thinness and complexity of neuronal structures, it is still extremely hard to extract connectomes from actual nerve tissue. First, one needs to acquire the data. Therefore brain tissue is cut into thin slices which are then imaged under electron microscopy (EM). Data processing and analysis is even more challenging. It requires the creation of 3D models of the cells by aligning the slices, the segmentation or identification of cellular compartments and the extraction of the connections between them. To overcome the challenges, combinations of visual computing, machine learning and human annotation are used in our lab. This allows 100-fold faster reconstructions of neuronal circuits than methods without semi-automated reconstruction workflow (Berning et al., 2015).

1.2 Local Alignment

One particular challenge in the extraction of connectomes is the alignment of images. Warping artifacts and gradual shifts during EM make it hard to align the sub-images and slices of a whole dataset to generate the 3D models. The error is increasing with bigger datasets and at some point, it also becomes computationally very challenging to come to a global alignment (Beining, personal communication). The image alignment is often based on cross-correlation and has a runtime of approximately $O(N^4)$ (Boergens, personal communication). An approach that might solve this problem is to first align smaller sub-cubes, where warping does not have a strong impact and align neighboring sub-cubes independently, afterwards (Beining, personal communication). The overlap that is used for alignment of the sub-cubes is introduced by the imaging technique, which partitiones the recording of one slice into multiple sub-images with a certain overlap. This approach of sub-cube alignment will in the following be referred to as local alignment.
This thesis focuses on building important parts of the local alignment algorithm and getting an intuition of what the challenges and solutions are. Aligning neighboring sub-cubes is a challenging task. The objects in the overlap of neighboring sub-cubes can strongly differ due to distortions along the z-axis. The distortions result from the warping artifacts during electron microscopy and the transformations that were applied in order to align the slices in z to get the 3D image volume. Another challenge is the varying size of the overlap that can range from under 220 nm to 1700 nm. The smallest overlap contains nearly no information about the shape of a neural process that lies in there. Those distortions introduced by the data acquisition and processing come together with the very complex shapes of the neural processes, themselves. They have thin structures and can partially be of similar shape which makes a unique matching difficult.
Since for a connectome only the identity and connections of cells are of interest the matching does not necessarily need to be on the level of 3D image volumes but the same cells have to be matched correctly across sub-cubes. Ideally, this should be achieved fast, which is challenging because of the huge data set size, and with high accuracy because errors potentially accumulate throughout the whole data set.





(a) **(b)**

Figure 1.1: (a) "93 μm × 60 μm × 93 μm sized serial blockface scanning electron microscopy (Denk and Horstmann, 2004) dataset from mouse somatosensory cortex layer 4 (dataset 2012-09-28_ex145_07x2, K.M.B. and M.H., unpublished data). "Orientation with respect to pial surface (Pia) and white matter (WM) is indicated" (Berning et al., 2015). (b) Rendering of neurons from mouse cortex S1, layer 4 (Berning et al., 2015). The large round structures are somata, the thin structures with spines are dendrites and the thin strucures without spines are the outgoing axons.

On the way of solving this problem are two particular tasks that we want to work out. The first one is to implement and evaluate a simple baseline solution that is implemented on the image level and fulfills the above requirements as good as possible. A baseline implementation is needed to compare future approaches and get a feeling on what minimum accuracy is to expect. The second task is to work on the feature level and extract rich features that can later be used to match processes. There are many very similar cells which are hard to separate. Therefore, the features need to have strong discriminative power. It would be hard to match processes based on hand-crafted features because of their potentially lacking descriptive power. For this reason, features that are directly learned from the data will be introduced.

1.3 Content

This thesis focuses on building important parts of the local alignment algorithm and getting an intuition of what the challenges and solutions are. To do so, first, the basics will be covered in Chapter 2. They are split into two sections. Firstly, the background containing all the important information to understand the process of connectome extraction and to understand the motivation of this thesis is introduced. Secondly, the related work that covers the most important methods that need to be known in order to understand the newly introduced concepts is explained. The concepts will be outlined in Chapter 3. There the baseline implementation and the 3D shape descriptor will be explained. The latter is needed to extract features from 3D data. Chapter 4 follows with several experiments to evaluate the introduced concepts and Chapter 5 completes this thesis by giving a general discussion and insights in further work.

2 Background and Related Work

This chapter covers the biophysical background of connectomics and presents related work from the fields of machine learning, image registration and graph theory. Hereby, the first subsection summarizes basics of connectome extraction. The second subsection covers the most important methods that need to be known to understand newly introduced concepts of this thesis and discusses related approaches.

2.1 Background

In order to understand the motivation and concepts introduced in this thesis one needs to understand the processing pipeline used to create a connectome. The following sections explain the major steps of this pipeline: Data acquisition and alignment, segmentation and agglomeration.

2.1.1 Data Aquisition and Alignment

The human brain contains extremely thin neural structures at the scale of down to several nanometers. The wavelength of a visible light photon is larger than the structures to be observed and hence cannot reveal the needed details. EM (Knoll and Ruska, 1931) surpasses this problem by using a beam of accelerated electrons, which have a shorter wavelength than light, as a source of illumination (Bogner, 2007). This results in resolutions as small as 50 pm. The dataset that was used in this thesis was imaged with a resolution of 11.24 nm in x and y. Since tissue is nearly 'transparent' to electrons, it first needs to be stained with hard metals to scatter the electrons and thus increase the contrast between different structures (Watson, 1958). After staining the tissue it is imaged with Serial Block-Face Scanning Electron Microscopy (SBFSEM, Denk et al., 2004). In SBFSEM the surface of the tissue block is imaged and afterwards a thin slice of 28nm is cut off with a diamond knife. This process is repeated iteratively. Due to the limiting field of view of an electron microscope, each single slice is imaged by partitioning it into columns of 36 μm with a certain overlap. Those sub-images are registered to obtain the whole slice and all subsequent slices are then aligned to obtain the 3D volume. The aligned slices will be referred to as raw data. The approach that is discussed in this thesis, local alignment, bypasses the step of intra-slice registration and directly aligns each column of sub-images along the z-axis. This is done to reduce the accumulating distortions introduced by the imaging. Afterwards, the idea is to stitch together the sub-connectomes. However, the following steps are the same for both approaches.

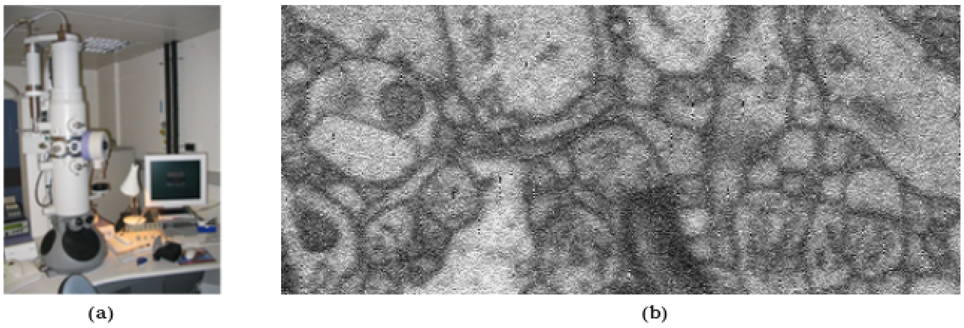


Figure 2.1: (a) Philips FEI Tecnai 12 Transmission Electron Microscope (David J Morgan, 2003, at flickr.com) (b) A part of one slice of the raw data. The data was acquired with EM from mouse whisker barrel cortex layer 2-4 (dataset 2017-02-14_st143_PC4_2048slices, Hua, unpublished data)

2.1.2 Segmentation

The generated raw data needs to be segmented to partition the intensity values of the raw data into semi-reasonable objects. SegEM (Berning et al, 2015) is a pipeline for semi-automatic segmentation of raw data. First, the raw data is partitioned into multiple cubes to parallelize the segmentation. For each cube, a convolutional neural network is used to classify a voxel to be either intra- or extracellular. The associated extracellular voxels define membranes, which surround a biological structure, e.g. a neural process. On this a watershed algorithm (Vincent and Soille, 1991) is applied to segment the cube. Since it is impossible to find a perfect segmentation, the parameters are chosen very tight to perform an over-segmentation. This means that the segmentation is biased towards split errors in order to minimize the number of merger errors. Split and merger errors denote segmentations that wrongly split one object into multiple or merge multiple objects into one. The resulting data structure is represented as a matrix with the same size as the raw data, where every entry corresponds to the voxel at the same position in the raw data and denotes which segment this voxel belongs to.

In contrast to the classical pipeline, the local alignment approach can lead to different segmentations for the same process due to the fact that the processes in the overlap are segmented two times with different context.

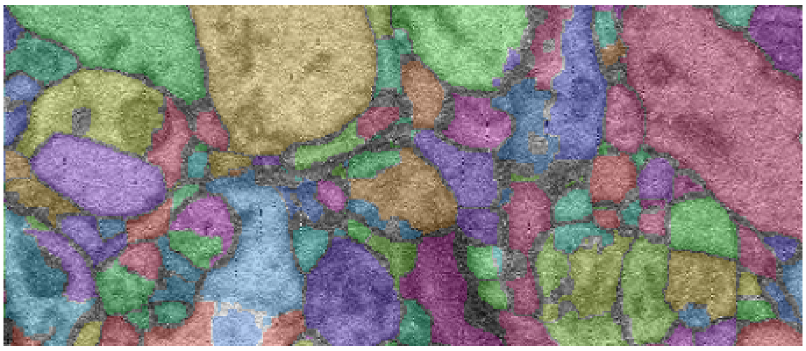


Figure 2.2: The segmentation laid over the raw data. Note that there are multiple splits due to the oversegmentation.

2.1.3 Agglomeration

A perfect segmentation contains neither split errors nor merger errors (Haralick et al., 1985). Hence the split errors introduced by the over-segmentation need to be corrected. Therefore, a neighborhood graph is calculated that assigns all neighboring segments a probability of being connected, i.e. belonging to the same process. This graph can then be used to connect neighboring segments and create agglomerates (Berning et al., personal communication). In the ideal case, one agglomerate corresponds to one process. However, in practice processes are sometimes merged together or split into multiple fractions.

Since the segmentations in the local alignment approach can be different for the same processes in the overlap, the agglomeration can also be different for the same processes.

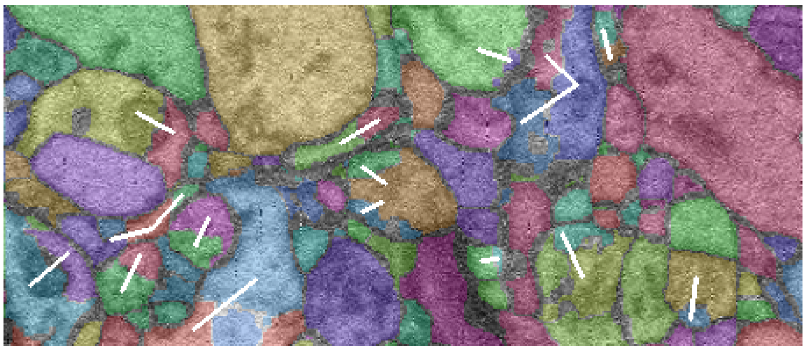


Figure 2.3: An illustration of the resulting agglomeration. The structures that were split into multiple segments are agglomerated as indicated by the white lines.

2.2 Related Work

The following subsection covers the most important methods that need to be known to understand the newly introduced concepts of this thesis. In particular, it discusses related approaches from the fields of machine learning, image registration and graph theory.

2.2.1 Autoencoder

An autoencoder is an artificial neural network used for unsupervised learning that learns to duplicate its input to its output. By introducing a hidden layer the network can learn a representation of the data. Typically, the representation of the data is smaller than the input and, thus can be used for dimension reduction (Hinten et al., 2006). The representation will in the following be referred to as encoding. An autoencoder consists of two parts, the encoder and the decoder. The encoder takes the input $\mathbf{x} \in \mathbb{R}^n = \mathbf{\Omega}$ and maps it to the encoding $\mathbf{z} \in \mathbb{R}^p = \mathbf{\Psi}$. The decoder maps the encoding $\mathbf{z}$ to the reconstructed input $\mathbf{x}' \in \mathbb{R}^n = \mathbf{\Omega}'$. When using a squared error (SE) as metric the encoder and decoder can be defined as transitions ϕ and φ, such that $\phi : \mathbf{\Omega} \rightarrow \mathbf{\Psi}$, $\varphi : \mathbf{\Psi} \rightarrow \mathbf{\Omega}'$ and $\phi, \varphi = \underset{\phi,\varphi}{\arg\min} \, ||\mathbf{\Omega} - (\phi \circ \varphi)\mathbf{\Omega}||^2$.

In the simple case of one hidden layer, the encoding is $\mathbf{z} = \sigma(\boldsymbol{W}\mathbf{x} + \mathbf{b})$ and the decoding is $\mathbf{x}' = \sigma'(\boldsymbol{W}'\mathbf{z} + \mathbf{b}')$. $\boldsymbol{W}$ is a weight matrix and $\boldsymbol{b}$ a bias. The reconstructed input $\mathbf{x}' \in \mathbb{R}^n = \mathbf{\Omega}'$ should be as close to the input $\mathbf{x} \in \mathbb{R}^n = \mathbf{\Omega}$ as possible, hence one can use the SE as a measure to estimate the difference between the input and the reconstruction.(adapted from Vincent et al., 2010) Autoencoders are trained to minimize such reconstruction errors,

$$\mathscr{L}(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||^2 = ||\mathbf{x} - \sigma'(\boldsymbol{W}'(\sigma(\boldsymbol{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')||^2.$$

A simple variation of the basic autoencoder described above is the denoising autoencoder (DAE). Instead of the input $\mathbf{x}$ the input of a denoising autoencoder is a corrupted version $\bar{\mathbf{x}}$ of $\mathbf{x}$. However, the DAE still minimizes the same reconstruction loss between the original $\mathbf{x}$ and its reconstruction $\mathbf{x}'$ which allows to extract representations $\mathbf{z}$ useful for denoising (Vincent et al., 2010).
A learned representation can be thought of as features of the learned object. This allows one to extract automatically learned features from the data. Wang et al. used this characteristic to extract learned features from 3D voxel data and showed that, for large datasets, these learned features are more distinctive than hand-crafted features. A simple autoencoder for dimension reduction is illustrated in the following figure:

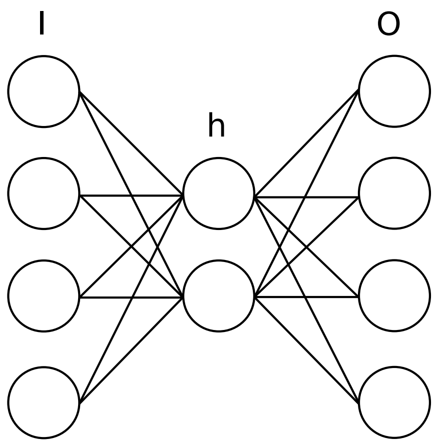


Figure 2.4: The structure of a simple autoencoder for dimension reduction. The hidden layer h is smaller than the input layer I and output layer O. The number of nodes is just an illustration the size of I equals the size of O.

2.2.2 Image Registration

Image registration is a common problem in many computer vision tasks such as automatic panoramic image stitching (Brown et al., 2006) or aligning multiple slices of brain tissue to one 3D volume (Beining, personal communication). It describes the process of transforming two images to the same coordinate system. This is typically done by aligning the same content in both images. The image that predetermines the coordinate system and hence is not transformed is referred to as the target. The image that is aligned to the target is referred to as the source. Figure 2.5 illustrates this process. Image registration methods can be devided into two classes. First, intensity based approaches and second, feature-based approaches (Teverovskiy et al., 2007). Intensity-based algorithms calculate metrics of overlaying pixels and iteratively try to minimize the metric. A prominent setting is to use the mean squared error (MSE) as a metric and gradient descent (GD) as an optimizer. The algorithm then tries to minimize the metric by following the gradient descent and in the end, the images should theoretically be registered.

Formally, let $\boldsymbol{\mu}$ be the external transformation parameters that describe the transformation from the source image i_s to the target image i_t. We aim to find the transformation T_μ with parameters $\boldsymbol{\mu}$ that define the transformation to register i_s to i_t, correctly. The problem of finding this transformation can be formulated as a minimization problem with

$$\mathscr{L}(\hat{\boldsymbol{\mu}}, i_t, i_s) = \arg\min_{\mu} \mathscr{L}(\boldsymbol{\mu}, i_t, i_s),$$

where the loss $\mathscr{L}$ is defined by the used metric, and $\hat{\boldsymbol{\mu}}$ are the parameters that corrctly register the two images. The used metric is also called similarity measure. For clarification we will use the MSE metric and calculate the loss by

$$\mathscr{L}(\boldsymbol{\mu}, i_t, i_s) = \frac{1}{N} \sum_{k=1}^{N} \| i_{t,k} - T_\mu(i_{s,k}) \|^2,$$

where N denotes the number of pixels. In order to find the minimum loss $\mathscr{L}$ a GD algorithm is used. It finds the direction of the metric derivative at its actual point and takes a step towards that direction. Repeating this iteratively potentially finds the minimum of a function. α denotes the step size and t the time. The refined parameters can then be calculated by

$$\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t - \alpha \frac{\delta \mathscr{L}(\boldsymbol{\mu}, i_t, i_s)}{\delta \boldsymbol{\mu}}.$$

Feature based algorithms on the other hand try to extract distinctive points from the images and find transformations between those points. Typically, the points are additionally rated according to their distinctiveness to make sure that the transformation is calculated at the basis of the best points.

Furthermore, the transformation model has a strong impact on the registration. It defines what class of geometric transformation is applied to the source. Naturally, the more complex the transformation model is the more degrees of freedom exist and the more complex is the registration. However, a more complex transformation model allows aligning stronger misalignments and distortions.

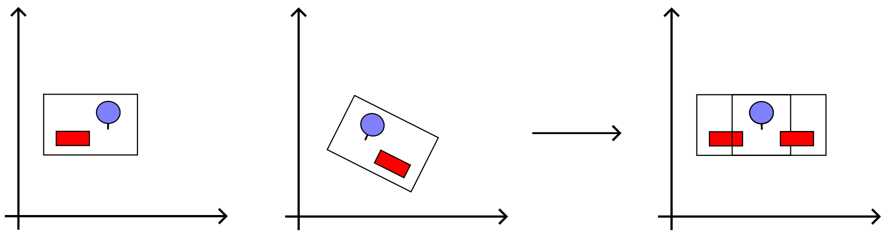


Figure 2.5: Two images with different orientations that are registered onto the same coordinate system. Left is the target image, in the middle the source image and right the result. Note that the position of the target image remains unchanged.

2.2.3 PointNet

PointNet is a neural network architecture that works directly on point cloud data. Qi et al. showed that it can keep pace with state of the art techniques for object classification (Qi et al., 2016). Point cloud data has the disadvantage that it has no order which makes it hard to handle for regular network architectures. PointNet solves this problem by approximating a general function defined on a point set by applying a symmetric function on transformed elements in the set, i.e. the point cloud, by

$$f(\mathbf{x}_1, ..., \mathbf{x}_n) \approx g(h(\mathbf{x}_1), ..., h(\mathbf{x}_n)),$$

where $f : 2^{\mathbb{R}^N} \to \mathbb{R}$, $h : \mathbb{R}^N \to \mathbb{R}^K$ and $g : \underbrace{\mathbb{R}^K \times \cdots \times \mathbb{R}^K}_{n} \to \mathbb{R}$ is a symmetric function. In practice h is approximated by multiple convolutional layers and g by a max pool function (Qi et al., 2016). In Figure 2.6 the invariance to permutation is clarified.

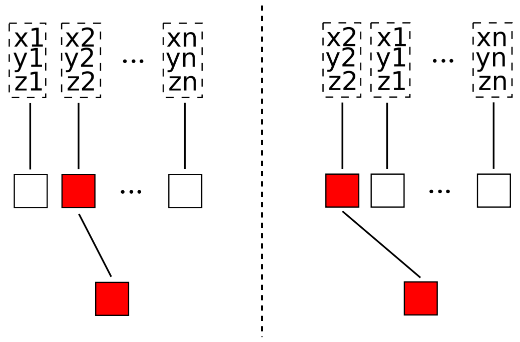


Figure 2.6: An illustration of how PointNet handles permutations. Left and right show the same point cloud with different permutations. On both the same convolution is applied with a kernel of size (3x1). The resulting vector is visualized by the squares. The red square indicates the maximum value. On this vector, a max pool operation is applied. Both permutations result in the same value.

2.2.4 Hungarian Algorithm

A weighted complete balanced bipartite graph is a weighted graph whose vertices can be partitioned into two disjoint sets S, T with $|S| = |T|$ such that no edge $e \in E$ connects two vertices of the same set and where every vertex in S is connected to every vertex in T. For such a graph a perfect matching is defined which matches all vertices of the graph to minimize the cost, i.e. the weight of the edges. An algorithm that solves this problem in polynomial runtime is the Hungarian algorithm (Kuhn, 1955). Given a weighted complete balanced bipartite graph $G = (S, T, E)$ with nonnegative cost $c(ij)$. We want to find a matching from T to S which minimizes the cost. The minimum cost of the perfect matching equals the maximum of a weighted-covering of c. A weighted-covering is a function $\pi : S \cup T \rightarrow R$ for which $\pi(i) + \pi(j) \leq c(ij)$ for every edge $ij \in E$ and with value

$$\pi = \sum_{v \in S \cup T} \pi(v) \text{ (adapted from Egerváry, 1931).}$$

An edge ij is called tight if $\pi(i) + \pi(j) = c(i, j)$. Let G_π be the subgraph of all tight edges in G and M a matching in G_π. Every element of M is orientated from T to S while all other edges in G_π are orientated from S to T. Let $R_S \subseteq S$ and $R_T \subseteq T$ be the vertices not covered by M, i.e. those orientated from S to T. Let Z be the set of vertices reachable in G_π from R_S by a directed path. Initally, π is 0 everywhere and M is empty. Hence only the vertices that are reached from an edge with cost 0 are in Z. If $R_T \cap Z$ is non-empty, then reverse the orientation of one path to increase the size of the resulting matching M' by 1. The process is repeated with matching M'. If $R_T \cap Z$ is empty, let $\Delta := \min\{c(i, j) - y(i) - y(j) : i \in Z \cap S, j \in T \setminus Z\}$ Increase π by Δ on all vertices $v_i \in Z \cap S$ and decrease π by Δ on all vertices $v_i \in Z \cap T$. The resulting π' is also a weighted-covering and thus, can be used to repeat the procedure with $G_{\pi'}$. Those steps are repeated until M is a perfect matching which gives the minimum cost. (Frank, 2004)

3 Concepts

This chapter presents the new concepts that were created as foundations for the local alignment algorithm. First, the baseline implementation will be explained in detail and secondly, the 3D shape descriptor is introduced. The 3D shape descriptor is needed to extract features from 3D data.

3.1 Baseline Implementation

In order to compare future approaches and evaluate their usefulness, it is necessary to provide a baseline implementation. A simple but still meaningful approach that will be used as a baseline is the registration of corresponding image slices and the extraction of the resulting amount of overlap of segments.

3.1.1 Local Alignment Using 2D Image Registration

As described in Chapter 2 the 3D volume is generated by aligning consecutive slices along the z-axis. As the two sub-cubes are processed independently, they have slightly different alignments. Those result in differently shaped 3D volumes, which make it harder to match corresponding neural processes across the sub-cubes. For this reason only 2D slices are aligned instead of 3D volumes. This removes the introduced different shapes and results in higher performance. We know which slides to align across the sub-cubes by using the z coordinate introduced by SBFSEM. Sub-images in the same z plane have to be aligned. Briefly, the algorithm iteratively registers fractions of every slice and counts the overlap size of all segments. The segments with the most overlap will be matched together.

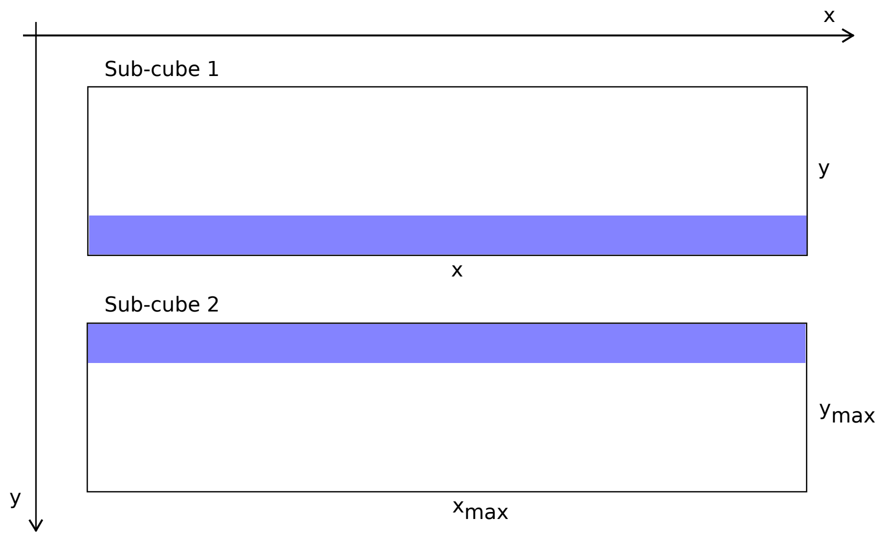


Figure 3.1: An illustration of two neighboring subcubes, their orientation and their overlap (in blue). The z axis is orthogonal to the sketch.

Following, a detailed enumeration of the steps of the algorithm is listed:

1. **Initialization** The first two views, $v_{1,1}$ and $v_{2,1}$, from the two neighboring sub-cubes have to be picked manually. It is necessary that the maximum overlap is used given a fixed x size for the source view $v_{2,1}$. This can be verified by the result of the registration. Let $r_{i,j}$ denote the transformed view from sub-cube i at time j. Since the views $v_{1,j}$ are the target views and hence are not transformed it applies that $r_{1,j} = v_{1,j}$. Additionally, let $r_{i,j,\{x|y\}_{\{min|max\}}}$ denote the respective border of the transformed view. The orientation is given by Figure 3.1.

Claim: Assumed $v_{2,j,x_{min}}$ and $v_{1,j,x_{max}}$ are outside of their respective sub-cube. For a source image with a given size in x $size_x$ the maximum overlap is used if

$$r_{2,j,y_{min}} \geq r_{1,j,y_{min}} \land r_{2,j,y_{max}} \leq r_{1,j,y_{max}} \land r_{2,j,x_{min}} \geq r_{1,j,x_{min}} \land r_{2,j,x_{max}} \geq r_{1,j,x_{max}}.$$

Borders are defined for the x position of the y borders and the y position of the x borders.

Proof: The four conditions correspond to the 4 borders. We will prove every condition isolated through contradiction.

$r_{2,j,y_{min}} \geq r_{1,j,y_{min}}$:	If $r_{2,j,y_{min}} < r_{1,j,y_{min}}$ then $r_{1,j,y_{min}}$ could be adjusted to increase the overlap. This doesn't violate any of the assumptions.
$r_{2,j,y_{max}} \leq r_{1,j,y_{max}}$:	If $r_{2,j,y_{max}} > r_{1,j,y_{max}}$ then $r_{1,j,y_{max}}$ could be adjusted to increase the overlap. This doesn't violate any of the assumptions.
$r_{2,j,x_{min}} \geq r_{1,j,x_{min}}$:	$r_{2,j,x_{min}}$ is exactly the end of the dataset and hence fix because of our assumption that $v_{2,j,x_{min}}$ is outside of its sub-cube. If $r_{2,j,x_{min}} < r_{1,j,x_{min}}$ then $r_{1,j,x_{min}}$ could be adjusted to increase the overlap. This doesn't violate any of the assumptions.
$r_{2,j,x_{max}} \geq r_{1,j,x_{max}}$:	$r_{1,j,x_{max}}$ is exactly the end of the dataset and hence fix because of our assumption that $v_{1,j,x_{max}}$ is outside of its sub-cube. If $r_{2,j,x_{max}} < r_{1,j,x_{max}}$ then $r_{2,j,x_{max}}$ could be adjusted to increase the overlap. This doesn't violate any of the assumptions.

In Figure 3.2 this is the case, hence the complete overlap was used. However, the target should not be much larger than the source since this complicates the registration. Additionally one can choose the x size of the source views, the x padding of the target views, the y padding of the target views and the number of slices that are skipped when taking the next slice. The idea of skipping slices is that one slice can be sufficient to match a whole process. Hence when a process appears in multiple slices only several of these will be registered to gain performance. The algorithm has a running time of $O(n)$ so by taking every third plane the runtime can by shortened by a factor of 3.

2. **Registration** The first step of finding matching processes in the two neighboring sub-cubes is to register corresponding views in the same slices. Corresponding views are further discussed in *Next View*. The slices can either be represented as raw images or as segmented images. Using the segmented images has the advantage of lowering the necessary amount of read requests since one needs to read only the segmented data but not the raw data. On the other hand, it has much less information and matching segments can differ strongly due to image warping during EM imaging. To register corresponding sub-images (views) an intensity-based approach and a point feature matching approach were used. A small experiment, to evaluate which representation and image registration algorithm works best, can be found in Chapter 4.1.1. The best results were accomplished by using the raw data and a point feature matching algorithm. Views containing mainly large processes (e.g. somata) tend to get aligned wrongly because of incorrectly found features in the homogeneous regions. Due to the prerequisite that the rotation and scale between the same slices are limited all matched points that would lead to a transformation that is rotated more than 12° and scaled more than 30% were removed.
Another problem are shifts and distortions in the same slice which make it impossible to correctly register two slices with a similarity geometric transformation. To minimize those effects, only views of size 300px were used. They can be sufficiently registered by using a similarity geometric transformation but still have enough information to find matching features. The found transformation is applied to the overlaying segmentation in order to register the segments. If no transformation can be found, the current slice will be skipped. This strategy can be refined by skipping only views instead of slices.

3. **Count Overlap** After applying the transformation the algorithm simply iterates over each pixel in the overlap and counts how often every segment ID pair overlaps. This information can later be summed up over all views to get the most likely match between pairs of segments.

4. **Next View** When a transformation is found and the overlapping segment pixels are counted the algorithm needs to calculate the next views of both sub-cubes to repeat the previous steps. Briefly summarized the algorithm first registers iteratively a whole slice and afterward goes back to the first view of that slice and, from that point, n steps up to the next slice. To illustrate this in detail, one can distinguish between two cases. First, when the next view lies within the same slice and second when the next view does not. For both cases, the algorithm takes advantage of the fact that the distortions in the slice are locally small and hence the size of the overlap in the y-direction is in the next view similar to the size in the actual view.

Let $ol_{j,\{x|y\}_{\{min|max\}}}$ denote the respective borders of the overlap of the two views after registration in time j. Furthermore, let pad_x, pad_y and $range_x$ denote the padding of the target views and the x size of the source views that were chosen in step 1. Let $c_{i,j}$ denote the nearest x border, given the y borders at time j that is still outside of the sub-cube i.
For case one the new bounding boxes are calculated by

$$\begin{aligned}
v_{1,t+1,y_{min}} &= ol_{t,y_{max}} - pad_x, & v_{2,t+1,y_{min}} &= v_{2,t,y_{max}},\\
v_{1,t+1,y_{max}} &= ol_{t,y_{max}} + range_x + pad_x, & v_{2,t+1,y_{max}} &= v_{2,t,y_{max}} + range_x,\\
v_{1,t+1,x_{min}} &= c_{1,t+1}, & v_{2,t+1,x_{min}} &= c_{2,t+1} - ol_{t,y_{max}} - ol_{t,y_{min}} - pad_y,\\
v_{1,t+1,x_{max}} &= c_{1,t+1} + ol_{t,y_{max}} - ol_{t,y_{min}} + pad_y, & v_{2,t+1,x_{max}} &= c_{2,t+1}.
\end{aligned}$$

For case two the new bounding boxes are calculated by

$$\begin{aligned}
v_{1,t+1,y_{min}} &= ol_{t,y_{max}} - pad_x, & v_{2,t+1,y_{min}} &= v_{2,t,y_{min}},\\
v_{1,t+1,y_{max}} &= ol_{t,y_{max}} + pad_x, & v_{2,t+1,y_{max}} &= v_{2,t,y_{max}},\\
v_{1,t+1,x_{min}} &= c_{1,t+1}, & v_{2,t+1,x_{min}} &= c_{2,t+1} - ol_{t,y_{max}} - ol_{t,y_{min}} - pad_y,\\
v_{1,t+1,x_{max}} &= c_{1,t+1} + ol_{t,y_{max}} - ol_{t,y_{min}} + pad_y, & v_{2,t+1,x_{max}} &= c_{2,t+1}.
\end{aligned}$$

5. **Get Maximum Correspondences** When every view was processed the algorithm evaluates which segment ID pairs overlapped the most often and labels those as the match. The result can even be interpreted in terms of a probability. The larger a segment in the first sub-cube and the higher the proportion of it overlaps with another segment in the second sub-cube the more likely it is that the match is correct, whereas very small segments with unique overlap or big segments with overlaps distributed among multiple segments can easily be wrongly matched.

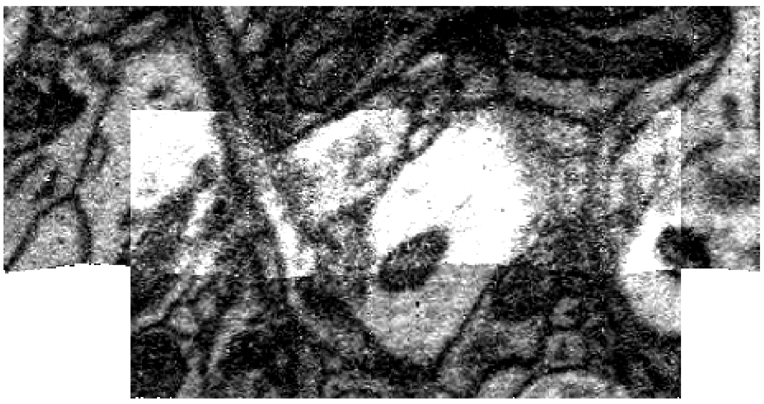


Figure 3.2: The two initially chosen views and the result of the registration. Note that the maximum overlap for the source is used. All edges fulfill their requirements described in 3.1.1.

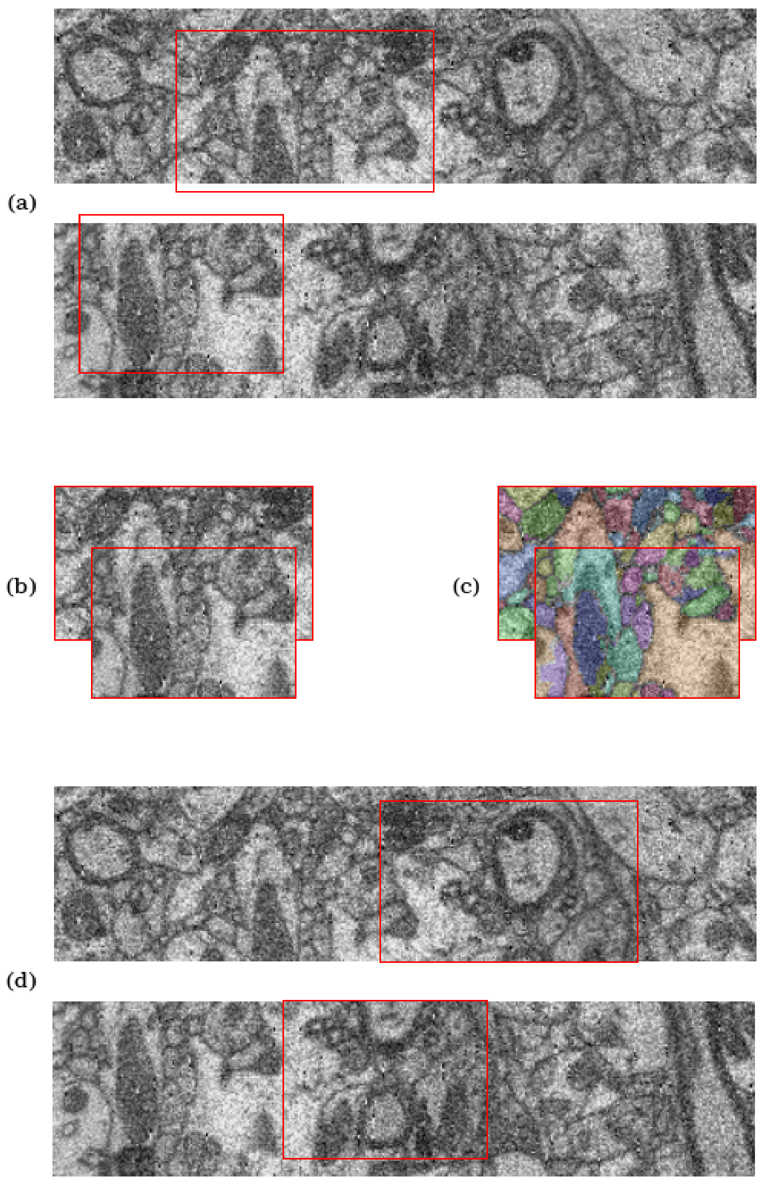


Figure 3.3: A flowchart illustrating steps 1 to 4. (a) The initial views are manually chosen. (b) The raw images are registered. (c) The same transformation is applied to the segmentation and the overlapping segment IDs are counted. (d) The next views are calculated and the next iteration starts. In the very end all counts are added to determine which segments match.

3.2 3D Shape Descriptor

This section covers the extraction of features from 3D models. 3D models are a suitable representation of agglomerates because agglomerates are graphs that connect multiple 3D volumes, e.g. segments. Obviously, in the first step of matching two agglomerates, one needs to find expressive features. Those features should be chosen by considering the prerequisites and the structure of the matching problem.

1. **Rotation Invariance** As described in Chapter 3.1.1 two sub-cubes can be differently rotated. However, the features of an agglomerate should remain the same no matter how it is rotated.

2. **Translations Invariance** Since the agglomerates are strongly shifted the features should be invariant to this.

3. **Deformation Sensitive** Unlikely many other shape descriptors, the features for the matching problem need to be very sensitive to deformations because the shape of a cell is primarily defined by its deformations.

4. **Expressive and Discriminative** Some agglomerates have just very few differences and still need to be distinguishable. Hence, the features need to capture as much shape information as possible (Xie et al., 2016).

Hand-crafted 3D shape descriptors are often not sufficiently robust and cannot deal with structural variations in the model (Fang et al., 2015). Additionally, they are biased towards specific properties that are not necessarily helpful for the matching problem (Qi et al., 2016) and time consuming to extract which makes them unusable for large-scale datasets (Wang et al., 2016). For this reason further investigation is done for approaches that automatically learn the features that are needed for a specific problem.
Leng et al. have shown that 3D convolutional autoencoders are well suited for learned 3D feature extraction and that the desired properties hold. Especially their abilities to directly learn shapes and their receptive field are advantages for good feature extraction of 3D objects. However, they become computational inefficient for larger objects and their dense data representation consumes a lot of memory. This makes it impracticable to use them for the high resolution data of connectomes. Intuitively, the representation of 3D objects in data that can be used by CNNs is very dense. The 3D object is rasterized and represented as a matrix where every point that is inside the object is 1 and every other point is 0. The points that are not on the surface of the object but on the inside are redundant information and hence can be removed. The same is true for the points that are outside of the object. So the real shape of the object is entirely encoded in its surface. Additionally, a sparse sampling of the surface is sufficient to interpolate the overall shape. A network architecture that can work on such sparse representations will be capable of processing larger and more detailed structures. One representation that fulfills these requirements are point clouds. Instead of discrete matrix representations, just the very position of points sampled from the surface of the object is stored. Furthermore, one can approximate large homogeneous structures with only a few points and important details with more. This allows weighting extraordinary features more than indiscriminative ones. Figure 4.5 shows that a cell of the size 100x80x30 voxels can be nicely represented with only 1024 x,y,z-points. A matrix of the same size, in contrast, has 240,000 entries. This means that a neural network working on point clouds instead of matrices could have only 3072 inputs instead of 240,000. This should be a huge gain in performance and is the motivation for developing a 3D shape descriptor that directly consumes point cloud data, in this thesis.

3.2.1 PointNet Autoencoder

A well-suited solution to fulfill the described requirements is an autoencoder that uses point clouds as input and output. Autoencoders are well known for extracting expressive features and the sparsity of a point cloud allows to capture much information without using much memory. Therefore this approach avoids the problems introduced by 3D convolutional neural networks but maintains the advantages of an unsupervised feature learning algorithm. A point cloud can be interpreted as an unordered set of points. Classic neural networks take ordered lists as input which makes them inappropriate to use. To solve this problem Qi et al. introduced PointNet, a neural network architecture that "directly consumes unordered point sets as inputs" (Qi et al., 2016). Using this architecture enables one to implement an autoencoder that works directly on point clouds.

Architecture

PointNet Autoencoder consists of two parts: The encoder and the decoder. The encoder maps the input, a point cloud, to the encoding, a vector. The decoder maps the encoding to the output, a point cloud that is equivalent to the input point cloud. Equivalent in this context means that the set difference of the input and output is empty. To guarantee that different permutations of point clouds that obviously define the same 3D volume are mapped to the same encoding an architecture similar to the architecture introduced in PointNet is used for the encoder. Since the permutation invariance is guaranteed in the encoding the decoder can be a simple fully connected network. The complete architecture can be seen in Figure 3.4.

Rotation invariance is achieved by rotating the input and mapping it to the same output. This way the autoencoder learns to extract the same features from different orientations. Translation invariance is achieved by transforming every object to the same origin. The sensitivity to deformations is automatically given by the different point clouds. Expressive and discriminative features cannot be guaranteed but will later be evaluated in experiments. Nevertheless, previous work showed that autoencoders are excellent to extract meaningful features (Wang et al., 2016).

To obtain a more robust model, the PointNet Autoencoder will be trained on jittered data to function as a denoising autoencoder. This, additionally, removes the impact of artifacts in the 3D volume.

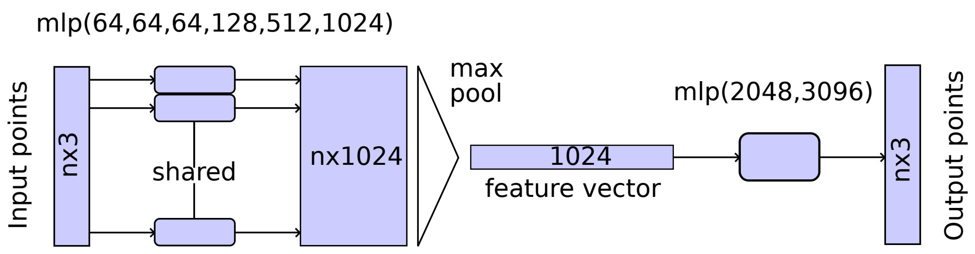


Figure 3.4: The autoencoder takes n points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is a reconstruction of the input point cloud. 'mlp' stands for multi-layer perceptron, numbers in bracket are layer sizes. Batchnorm is used for all layers with ReLU (adapted from Qi et al., 2016).

Loss Function

Traditional autoencoders use a simple squared error to calculate the difference between input, the target, and output. This naive approach is not suited for PointNet Autoencoder. If one imagines two times the same point cloud with different permutations $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$ such that $\boldsymbol{p_1} \neq \boldsymbol{p_2}$. Due to the permutation invariance of the encoder both are mapped to the same encoding $\boldsymbol{enc}$ which is desired behavior. Since the function learned by the neural network is deterministic this implies that the decoding $\boldsymbol{dec}$ of both encodings $\boldsymbol{enc}$ have to be the same, considering the order of the points. When now calculating the squared error between the two identical outputs and the two different inputs it is impossible to reach a loss of zero. The autoencoder tries to learn contradictory rules. Let the encoder and decoder be defined as transitions ϕ and φ. Additionally let $\boldsymbol{x}$ denote a vector of all inputs $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$, and $\boldsymbol{x'}$ a vector of all outputs $\boldsymbol{dec}$ and $\boldsymbol{dec}$. The encodings and the decodings are then calculated by

$$\phi : \boldsymbol{p_1} \to \boldsymbol{enc},$$
$$\phi : \boldsymbol{p_2} \to \boldsymbol{enc},$$

$$\varphi : \boldsymbol{enc} \to \boldsymbol{dec},$$
$$\varphi : \boldsymbol{enc} \to \boldsymbol{dec}.$$

The error $\mathscr{L}(\boldsymbol{x},\boldsymbol{x'})$ is then calculated with $\mathscr{L}(\boldsymbol{x},\boldsymbol{x'}) = \frac{1}{2}||\boldsymbol{x}-\boldsymbol{x'}||^2 = \frac{1}{2}(||\boldsymbol{p_1}-\boldsymbol{dec}||^2 + ||\boldsymbol{p_2}-\boldsymbol{dec}||^2)$.

Claim: Given three vectors $\boldsymbol{x},\boldsymbol{y},\boldsymbol{z} \in \mathbb{R}^n$ and an arbitrary metric $||d||$, then

$$\boldsymbol{x} \neq \boldsymbol{y} \Rightarrow ||\boldsymbol{x}-\boldsymbol{z}|| + ||\boldsymbol{y}-\boldsymbol{z}|| > 0.$$

Proof: The claim will be proven via contradiction.
Let $||\boldsymbol{x}-\boldsymbol{z}|| + ||\boldsymbol{y}-\boldsymbol{z}|| \leq 0$. Due to the non-negativity of the metric the sum of the summands $||\boldsymbol{x}-\boldsymbol{z}||$ and $||\boldsymbol{y}-\boldsymbol{z}||$ must be greater or equal to 0. With respect to our assumption that $||\boldsymbol{x}-\boldsymbol{z}|| + ||\boldsymbol{y}-\boldsymbol{z}|| \leq 0$ this means that $||\boldsymbol{x}-\boldsymbol{z}|| + ||\boldsymbol{y}-\boldsymbol{z}|| = 0$. This holds iff $||\boldsymbol{x}-\boldsymbol{z}|| = 0$ and $||\boldsymbol{y}-\boldsymbol{z}|| = 0$. This implies that $\boldsymbol{x} = \boldsymbol{z}$ and $\boldsymbol{z} = \boldsymbol{y}$ which results in $\boldsymbol{x} = \boldsymbol{y}$ because of the transitive relation. This is contradicting to the assumption that $\boldsymbol{x} \neq \boldsymbol{y}$ and hence the claim is proven.

Since $\boldsymbol{p_1} \neq \boldsymbol{p_2} \Rightarrow \|\boldsymbol{p_1} - \boldsymbol{dec}\| + \|\boldsymbol{p_2} - \boldsymbol{dec}\| > 0$, it holds true that

$$\begin{aligned}
\|\boldsymbol{p_1} - \boldsymbol{dec}\| + \|\boldsymbol{p_2} - \boldsymbol{dec}\| &> 0, \\
\|\boldsymbol{p_1} - \boldsymbol{dec}\|^2 + \|\boldsymbol{p_2} - \boldsymbol{dec}\|^2 &> 0, \\
\frac{1}{2}(\|\boldsymbol{p_1} - \boldsymbol{dec}\|^2 + \|\boldsymbol{p_2} - \boldsymbol{dec}\|^2) &> 0, \\
\frac{1}{2}(\|\boldsymbol{p_1} - \boldsymbol{dec}\|^2 + \|\boldsymbol{p_2} - \boldsymbol{dec}\|^2) &\neq 0.
\end{aligned}$$

Intuitively the target can be seen as the desired output. The autoencoder tries to adjust its parameters to come closer to the target. If it has now two different targets for the same encoding it does not know which target to approximate and tries to approximate both which results in contradictions. This motivates us to introduce a more complex loss function that resolves this problem. A loss function that is not dependent on the ordering of the output and that approximates the target with minimum effort. The simplest case of such a function is to calculate the squared error for every possible permutation of the target and using the permutation that results in the minimum value

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{x'}) = \tfrac{1}{n} \sum_{k=1}^{n} \underset{perm\ x_k}{\arg\min}\, \|\boldsymbol{x_k} - \boldsymbol{x'_k}\|^2.$$

In the previous example, this would lead to the same loss for both the different inputs and hence the autoencoder can minimize this loss until the target point cloud and the output point cloud are equivalent. The desired result would be achieved. For the rare case that multiple solutions are found an ordering can be defined but this case is negligible in practice. The downside to this approach is its $O(n!)$ runtime which makes it unusable for any application.
A more sophisticated solution introduced in this thesis is to interpret the problem as a graph matching problem. The target and output point clouds can be seen as two disjoint and independent sets X and Y where every point is one vertex. Every vertex in X has one edge to every vertex in Y with a weight that equals the Euclidian distance between the two corresponding points. Hence the two point clouds are now expressed as a weighted bipartite graph. This process is illustrated in Figure 3.5. For such a graph a minimum weight perfect matching is defined. When interpreting the weights as costs it finds the optimal matching to minimize the overall cost. Hence we find the matching of points that minimize the overall distance of the two point clouds. This is the same distance that would be calculated by the naive brute force approach so one can use the permutation of this matching as the basis of a loss function. An algorithm that solves this problem in polynomial time is the Hungarian algorithm.
Formally, define a bipartite graph by $G = (U, V, E)$ with partitions U and V, and edges E. Furthermore, let S and T denote the sets of points contained in point cloud A and point cloud B. Let E' be the set of all edges that connect a point $a \in S$ to a point $b \in T$ with weights that correspond to the euclidean distance $d = ||a - b||$. A bipartite graph can now be constructed from two pointclouds A and B by setting $U = S$, $V = T$ and $E = E'$ which results in $G = (S, T, E')$. If $|S| = |T|$ the graph is called balanced bipartide graph.
Our practice has shown that the Hungarian algorithm is too slow for large point clouds which is why additionally an approximation is presented. The permutation that results in the minimal distance is approximated by ordering every point from point cloud A and point cloud B by their z-coordinate. This is obviously not a very good approximation but its accuracy increases with sparser point clouds. The experiments also showed that this approximation extracts good features.

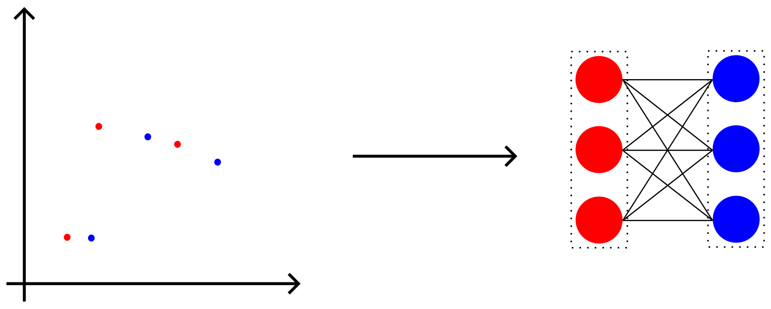


Figure 3.5: Two point clouds (red and blue) are transformed into a weighted bipartide graph. The weights are the distances between the two connected points.

4 Experiments

Experiments are divided into two parts: First, the experiments used in the context of the baseline implementation and second, the experiments in the context of the 3D shape descriptor.

4.1 Local Alignment Using 2D Image Registration

First the attempts to build the algorithm are summarized and next, the algorithm itself will be evaluated.

4.1.1 Registration

The following experiment evaluates which parameters and algorithms result in the best registration. Views are referred to as images, the image that preserves its transformation as the target, and the image that is transformed as the source. Size refers to the size in x direction except it is stated otherwise.

The homogenous and dense structures in brain tissue impair the registration. For this reason several different parameter and algorithms were tested and evaluated. In particular, motion models, image sizes, registration algorithms and image representations were evaluated.

Experiment

The motion model depends to a certain amount on the source image size. The smaller the source image the fewer distortions it has and hence a simpler motion model can be used to successfully register the image to its target. For efficiency reasons, a simple motion model and small source image size are preferred over a complex motion model. It is well known that the sub-cubes are translated and rotated hence at least a 'similarity' motion model is needed and was tested. Additionally, to determine if a more complex motion model improves the registration an 'affine' motion model was tested. For source image sizes 300px, 500px and 1000px were used. Intensity-based as well as feature-based algorithms were tested. The intensity-based algorithm uses the Euclidean distance as metric and gradient descent as optimizer. The feature-based algorithm extracts 'Speeded Up Robust Features' (Bay, 2006) and feature pairs that would lead to a transformation that is rotated more than 12° and scaled more than 30% were removed because they are likely a wrong match. Segments can either be registered by registering the raw image and applying the same transformation to the corresponding segmentation or by directly registering the binarized segmentation. The segmentation was binarized by setting all pixels that are 0, i.e. all border pixels, to 1 and all other pixels to 0. For the experiment different arrangements were used and evaluated by human expertise.

Result

This experiment showed that a 'similarity' geometric transformation in combination with a source image size of 300px is sufficient to register two images. Some example registrations can be seen in Figure 4.1. Figure 4.1 (a) shows the registration with a source image size of 1000px. The sides are very blurry which indicates a bad registration. Same goes for a source image size of 500px but not as massively. When using source images with a size of 300px the distortions seem to be no problem and the results are very good. Additionally, one can see in Figure 4.1 (c) and (d) that the feature-based algorithm yielded the better results. Figure 4.1 (e) shows that applying the feature-based algorithm on the binarized segmentation also results in good registrations. However, the registrations are not as robust as the registrations on the raw data. An affine motion model yielded no improvements.

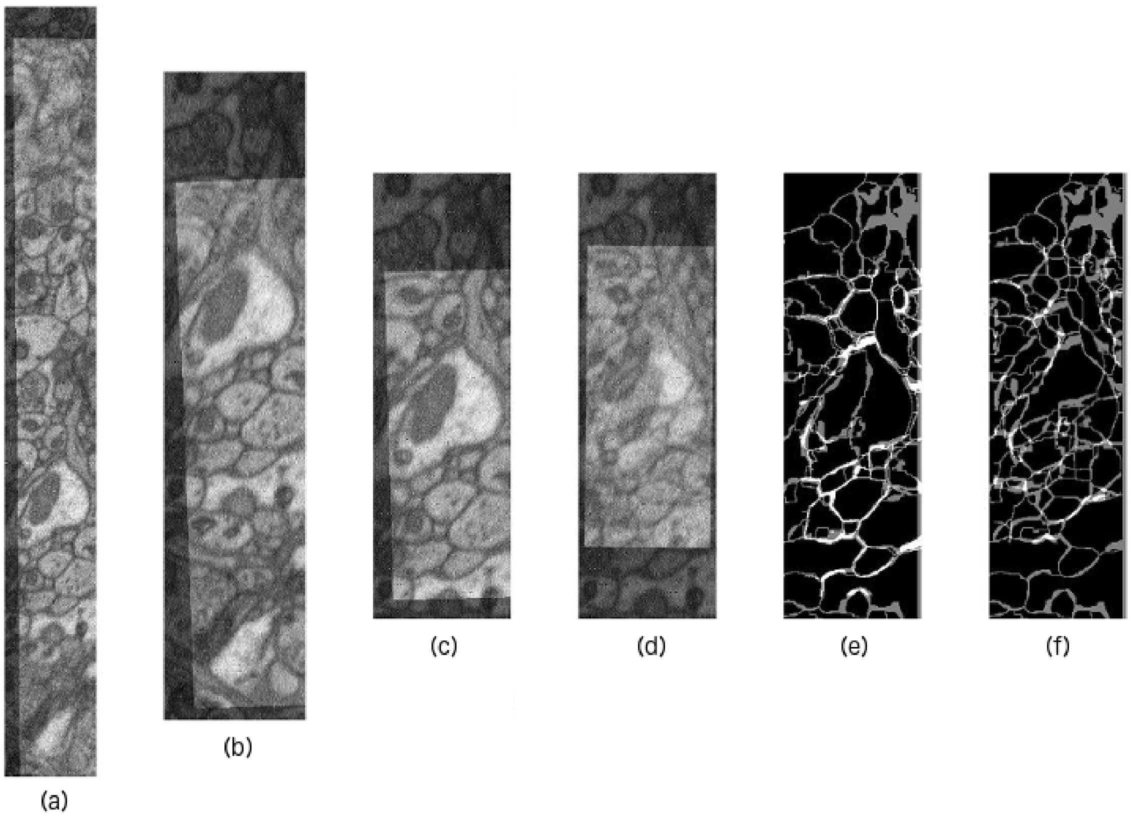


Figure 4.1: Example results of registrations with different settings: (a) source image size: 1000px, algorithm: feature-based, data: raw; (b) source image size: 500px, algorithm: feature-based, data: raw; (c) source image size: 300px, algorithm: feature-based, data: raw; (d) source image size: 300px, algorithm: intensity-based, data: segmentation; (e) source image size: 300px, algorithm: feature-based, data: segmentation; (f) source image size: 300px, algorithm: intensity-based, data: segmentation;

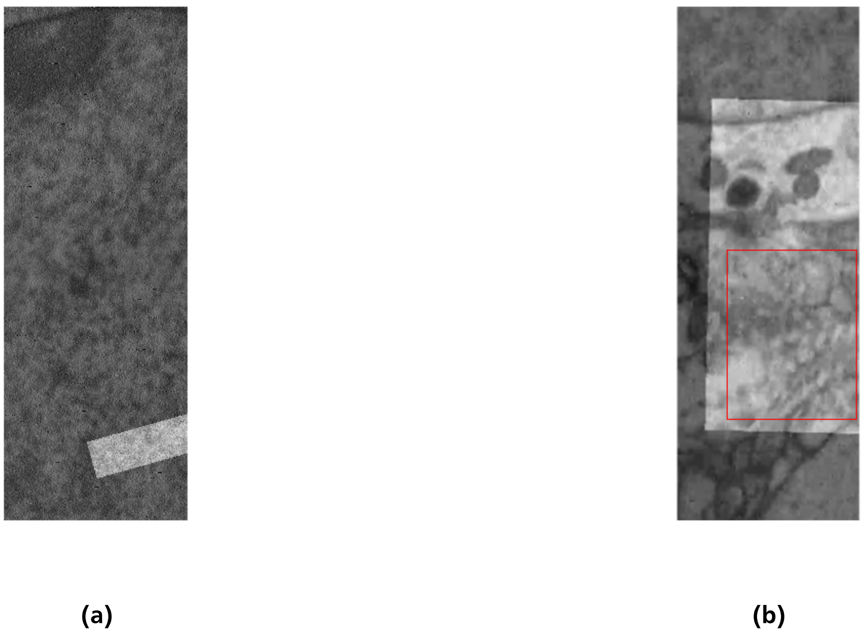


Figure 4.2: (a) Unsuccessful regsitration of two images inside a nucleus. (b) Partially unsuccessful registration of two images with small neural processes.

Discussion

The best registrations were achieved by using the raw data with source image sizes of 300px and a feature-based registration algorithm. These parameters were also used in later experiments. Using a 'similarity' geometric transformation was sufficient. However, due to the locally strong distortions, an 'elastic' transformation like a homeomorphism could yield even better results. The disadvantage of homeomorphisms is their longer runtime that accumulates when applied to multiple views. Due to the prerequisite that the rotation and scale differences between the same slices are limited, it is reasonable to remove all feature pairs that violate the predefined thresholds. Doing this removes very wrong feature pairs resulting in even better registrations. Using the raw data for registration instead of the binarized segmentation is preferred because of the different segmentations in the two sub-cubes. The segmentation can have different borders which make it more difficult to register the images. Using the raw data increases the robustness. One remaining problem is the lack of distinctive features in nuclei. Nuclei are large structures with very little texture. When the source image is completely inside the nucleus it is very hard to register even for humans. The registration of such an image without removing any feature pairs can be seen in Figure 4.2 (a). Such a registration is not meaningful. Nevertheless, this problem can be solved by detecting nuclei in advance and skipping all images inside nuclei. Afterwards, the nuclei can easily be matched with different algorithms due to their sparse distribution inside the cube and large size, or by interpolation of the transformation parameters from neighboring views.

4.1.2 Aligning a Block

In this experiment, the baseline implementation will be run on the segmentation of two sub-cubes.

Experiment

The dataset used for this experiment is mouse whisker barrel cortex layer 2-4 (dataset 2017-02-14_st143_PC3&4_2048slices, Hua, unpublished data) with a y-overlap between the two sub-cubes that ranges from 200nm to 1.5μm. The goal was to evaluate the accuracy of the approach. For evaluation 43 segments within a bounding box were manually matched in order to create a ground truth. Since the brain has many properties occuring very locally the matches were sampled in a relatively large bounding box (9μm × 1.5μm × 6μm) to reduce local influences. For registration a feature-based algorithm was used with a source image size of 300px.

Result

36 of the 43 annotated pairs were matched correctly. Hence, the algorithm achieved an accuracy of 0.84. The processing runtime of the algorithm for one 10μm slice (without the time for loading the data) was 14.2s. Although the measured runtime can vary it gives a brief estimate. The loading time, in contrast, was 340.7s. Errors occured more often in somata and in very thin processes that show strong distortions between the two sub-cubes as it can be seen in Figure 4.2.

Discussion

The resulting accuracy of 84% is better than initially expected. Although the runtime is just a brief estimate it can be seen that the algorithm is faster than loading the data which is sufficient in terms of speed. Additionally, the algorithm could be parallelized to speed it up. Errors occured more often in very thin processes that show strong distortions between the two sub-cubes as well as in somata, which contain very homogeneous structures without distinctive features. The first problem could be improved by using a more complex geometric transform, the second by detecting somata and skipping them in the registration process. Errors in the matching potentially propagate throughout the whole dataset and accumulate. Therefore the accuracy of the matching must be very high to obtain a good connectome in the end. Since obtaining a perfect matching will be very hard with this baseline approach, a function to measure the quality of a matching would be very useful. Such a function could be calculated by taking the size of the segment and the proportion of its overlap into account and normalizing the function values between 0 and 1. The larger the segment and the more unique its overlap the better is the match. In contrast, a very small segment with unique overlap or a large segment with overlaps distributed among multiple segments are likely to be wrongly matched. This function can be interpreted as probability of two segments being matched correctly. However the accuracy of this probability is biased towards larger and correct matches. A very small process that has no overlap with its true match will result in a probability value of 0, which is obviously a very bad accuracy. This function could also be used for a prematching to match the approximately 80% of the dataset. This prematching can then be used to match the remaining 'problematic' processes with more advanced approaches. The baseline approach can additionally be improved by taking more than one slice into account so mismatched in single slices are compensated. Another improvement can be obtained by using the agglomerated segmentation instead of the raw segmentation. This will remove many very small segments by merging them into larger agglomerates.

4.2 3D Shape Descriptor

This section illustrates the implementation of the autoencoder and its evaluation.

4.2.1 Simple PointNet Autoencoder

This section illustrates the process of building the autoencoder. The initial state was Qi's et al. PointNet architecture for supervised classification. Herefrom, we built an autoencoder that is invariant to permutations and nicely reconstructs the input. In order to reduce complexity, the problem was split into multiple steps: Firstly, a simple 'vanilla' architecture, secondly, a denoising autoencoder and thirdly, a sophisticated cost function were added. The first step referred to as 'simple autoencoder' includes just the minimum requirements for a not fully permutation invariant autoencoder. The input will not be modified and the output will just strive to reconstruct exactly the input. So there is no denoising and no permutation invariance in the output. However, the input should be invariant to permutations. For the encoding step, a slightly modified architecture from PointNet (Qi et al., 2016) was used (Figure 3.4, layers until 'feature vector'). It consists of multiple convolutional layers followed by a max pool function. For the decoding part, a naive first try was to reverse the encoding layer, which is an established practice (Xu et al., 2014). Next, instead of the reversed architecture, a fully connected architecture as it can be seen in Figure 3.4 was used for decoding. Both approaches were just trained on 2 samples with 1024 points. The idea was that if the autoencoder is not able to work on a very small dataset, it will also not be able to properly learn large data sets. The next step was to augment the autoencoder to work as a denoising autoencoder. Therefore the input data was randomly rotated by an angle α ∈[-12°,12°] and jittered. For jittering a normal distribution with σ value of 0.01 and 0.001 with a clip of 0.05 was tested. The output should still reconstruct the original unmodified point cloud.
The last step was to introduce a more sophisticated loss function that results in an autoencoder that is completely invariant to permutations. Therefore, the Hungarian algorithm was used to find the best matching of the reconstructed point cloud and the target point cloud. Afterwards, the target point cloud was permutated to correspond to the best matching and a squared error was calculated on the prediction and the permutated target.

Results

The reverted decoding architecture did not yield good results. The points tend to 'stick' to the axis and no reconstruction was done. The fully connected decoding architecture, on the other hand, yielded great results and could reconstruct the input almost perfectly. The reconstruction results can be seen in Figure 4.3. Denoising worked also quite good for the lower value $\sigma = 0.001$. For the larger value $\sigma = 0.01$ the jittering was too strong and the autoencoder got stuck in a local minimum. Using the Hungarian version of the loss function resulted in a perfect reconstruction, too. However, it converged slightly faster. A more detailed comparison between the two loss functions can be found in Section 4.2.3.

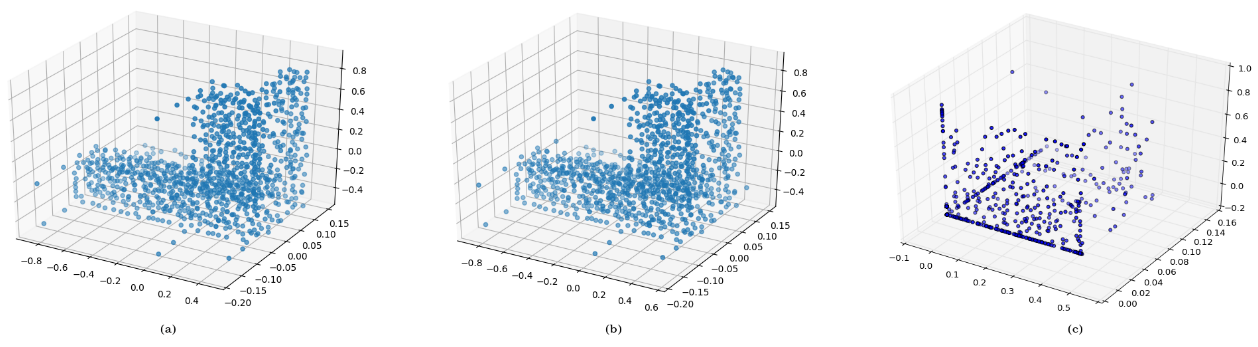


Figure 4.3: (a) The input point cloud of a bench. (b) The output pointcloud after 300 epochs of training from the fully connected decoding architecture. (c) The output pointcloud after 300 epochs of training from the reverted decoding architecture.

Discussion

The problem with the reverted encoding architecture was most likely due to the inverted max pooling function that introduced many zeros. Using the fully connected layers worked much better and resulted in a very good reconstruction. Using $\sigma = 0.01$ for jittering the point cloud seemed to be too high. Using a lower value and the rotation worked fine and helped to make the autoencoder more invariant to small artifacts and rotation (Qi et al., 2016). The cost function using the Hungarian algorithm helped the autoencoder to converge faster. This was due to the fact that now the points are not forced to match their respective index but their 'real' match. Intuitively this means that the points were less moved because they were matched smarter.

4.2.2 Pointnet Autoencoder Match Three Agglomerates

In order to get a first feedback if the features extracted by the 3D shape descriptor are sufficient to match cells in general, we tried to match a small set of agglomerates by just using the learned features. For this 3 matching agglomerates were traced in the overlap of both sub-cubes. Figure 4.4 shows one of the agglomerates in the x,y-plane of one sub-cube and the two corresponding 3D models from both sub-cubes. As it can be seen, they had equal shapes but still diverged in some details. For this reason they should have equal features. Matching agglomerates should be found by clustering the feature vectors of all agglomerates into 3 classes with 2 samples in each. The number of classes corresponds to the number of matches and every match contains 2 agglomerates. The clustering was done by calculating the Euclidean distance between all autoencoder feature vectors and matching those with the smallest distances. The network used for this experiment used 1024 points as input and thus 1024 points as output. As loss function only a MSE on the points ordered in z was used. The small number of samples made it unnecessary to use a more sophisticated loss function. Apart from that, the same architecture presented in Chapter 3.2 was used. For data preprocessing a random rotation by an angle $\alpha \in$[-12°,12°] and a jittering was used. This forced the autoencoder to learn an invariance to rotation and jittering since for all settings the same decoding should be reconstructed and hence all modifications should also result in the same encoding. Two variants of training were tested in this experiment. First, the autoencoder was trained on all 6 point clouds extracted from the 3 matching agglomerate pairs and then, only on 3 from one sub-cube. Afterwards, with the trained autoencoder the feature vectors from all agglomerates were extracted and clustered.

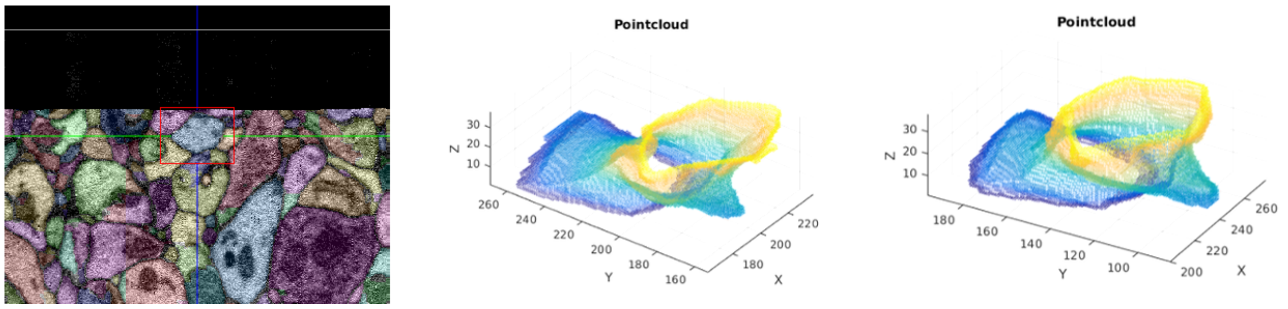


Figure 4.4: The point clouds from the same agglomerate from two sub-cubes. The raw data of one sub-cube can be seen in the left image. Note the differences in the two point clouds. The colors encode the z value and are placed for representation reasons.

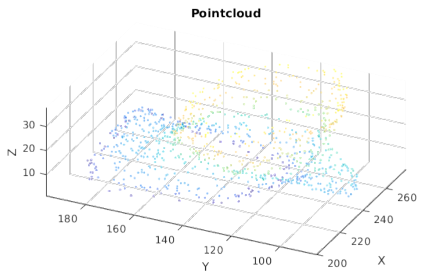


Figure 4.5: A point cloud from one of the agglomerates downsampled to 1024 points. The shape of the object is sufficiently defined. The colors encode the z value and are placed for representation reasons.

Results

All agglomerates could be matched by searching their nearest neighbor in the feature space. Figure 4.6 shows a table of the Euclidean distances between the feature vectors from their respective agglomerates. Receiving the agglomerates from the two sub-cubes was a nontrivial task. The small overlap made it hard to extract agglomerates that have sufficient size in z. Only vertically oriented agglomerates could lie completely within the overlap.

Distances			
	sub-cube1, agglo1	sub-cube1, agglo2	sub-cube1, agglo3
sub-cube2, agglo1	34.55	56.42	45.64
sub-cube2, agglo2	42.60	23.73	39.41
sub-cube2, agglo3	40.23	32.17	24.11
sub-cube2, agglo1	28.98	63.29	55.008
sub-cube2, agglo2	52.76	13.05	45.56
sub-cube2, agglo3	62.18	42.78	16.19

Figure 4.6: The resulting Euclidean distances of the feature vectors corresponding to the respective agglomerate after 500 epochs of training. The first three rows denote the distances when the autoencoder is trained on all agglomerates and the last three when it is only trained on the agglomerates of one sub-cube

Discussion

The goal of this experiment was to see if the approach can match agglomerates in a very simple setting. Indeed, the diagonal, e.g. the distances of the correct matches, yielded the smallest values. The results were even better when training the autoencoder on only one of the sub-cubes. However, in both settings, the autoencoder was strongly overfitted which is why the results should not be overinterpreted. A more serious problem was the nontrivial data acquisition. Only a few processes had sufficient volume within the overlap and were uncut along their transverse extent. Also, even if they have enough volume, it is necessary to determine the overlap of the sub-cubes without knowing which processes match together. Future datasets might have larger overlap but for the dataset used in this thesis a pure feature matching approach on the 3D data will not work. For this reason, one could start with the baseline approach to receive upper and lower bounds of the overlap and then use these boundaries to extract the 3D models within the overlap.

4.2.3 Pointnet Autoencoder Test Features

The goal of a 3D shape descriptor is to extract meaningful features. To evaluate if the autoencoder learns useful features it was tested how well it performs in a classification task. For this, the ModelNet40 dataset, which contains 12,311 CAD models with 40 class labels, was used. In the training process, the class labels were not used and the autoencoder just tried to reproduce its inputs. To additionally test how well the autoencoder generalizes only 9,843 of the 12,311 samples were used for training. In the testing process, all samples were transformed into their feature space using the autoencoder and the labels were appended to the respective feature vectors of the training set. Next, a k-nearest-neighbors classification with k=5 was done on the 2,468 unlabeled samples. The accuracy was then calculated by dividing the correctly classified samples by the total number of samples in the test set, i.e. 2,468. This experiment was firstly done with point clouds containing 128 points and the Hungarian version of the cost function and secondly, with point clouds containing 1024 points and the approximated version of the cost function. Jittering and rotation of samples was turned off for performance reasons. To illustrate the differences between the two loss functions, the simple version of the autoencoder was additionally trained on 128 points.

Results

The first setting with point clouds containing 128 points and using the Hungarian version of the cost function had an accuracy of 82.5%. The second setting with the point clouds containing 1024 points and using the approximated version of the cost function yielded an slightly increased accuracy of 83.1%.

When comparing the two loss functions in an equal setting the Hungarian version yielded better results. It converged faster, resulted in a better reconstruction and achieved higher accuracy. The simple version achieved an accuracy of 79.1% while the hungarian version achieved 82.5%. In Figure 4.7 the reconstruction error over the number of epochs of both versions can be seen.

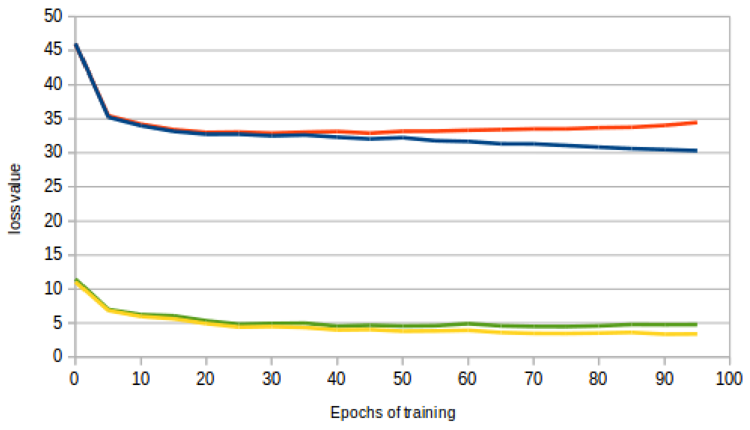


Figure 4.7: A comparison of the two loss functions on point clouds containing 128 points. The loss value is plotted over the number of epochs. (red) Loss value of the simple loss function on test set. (blue) Loss value of the simple loss function on train set. (green) Loss value of the Hungarian loss function on test set. (yellow) Loss value of the Hungarian loss function on train set.

Discussion

Both versions performed good. The original PointNet paper had an accuracy of 89.2% by training in a supervised setting while our autoencoder achieved an accuracy of 83.1% in an unsupervised setting. One can argue that the autoencoder learned to a certain degree useful features that can be used to describe the objects. The increased accuracy of the second approach with the approximated cost function was due to the increased number of points that resulted in a more detailed shape. When comparing both loss functions with an equal number of points the hungarian version outperformed the simple version. As stated in Chapter 4.2.1, this was due to the fact that the points were not forced to match their respective index but their 'real' match. Intuitively this means that the points were less moved because they are matched smarter. This results in both, a faster convergence and a lower reconstruction error. The downside is the much higher runtime of this approach. For this reason, for most applications the simple version of the loss function with more points is suitable. However, for applications with small point clouds or small datasets the hungarian version can be better suited.

5 Discussion and Outlook

In this thesis a pipeline for matching neural processes across sub-image volumes is presented. In particular an image registration approach that achieved an accuracy of 84% on real data and an unsupervised 3D shape descriptor that directly works on point cloud data are presented. The image registration approach matches agglomerates by registering their raw image data. It then counts the overlap and from this decides if two agglomerates match. The 3D shape descriptor is an extension of PointNet (Qi et al, 2016). It combines the ability of PointNet to work on point cloud data with an autoencoder and a cost function that guarantees invariance to permutations in the input. The unsupervised classification task on ModelNet40 showed that the shape descriptor is able to extract useful features. This bachelor thesis only covers some aspects of matching neural processes from two overlapping sub-cubes. Besides building the two described major parts, a better understanding of the overall problem and what its limitations are was gained. The three major limitations are the partially very small overlap, the partly strong distortions and the unequal aligning, which results in different 3D shapes of the same cells in different sub-cubes.
Since such volumes of imaged brain tissue consist of differently shaped neural processes that result in varying challenges there will not be one approach that matches all neural processes. There will rather be multiple approaches that each focus on different tasks. For this reason, a processing pipeline is needed for local alignment. Obviously, the more matches are found the easier are the remaining ones. This is why it would be advisable to start with the easiest (e.g. largest) neural processes and refine the matching until all processes are matched. In the following, a possible pipeline is illustrated that could be built as a next step from this thesis.
The easiest processes to match are somata, blood vessels and big dentrites. Somata, for example, can be extracted by their nuclei for which good heuristics exist in the lab. The matching is easy because of their size and sparsity. After matching the somata one has already first transformations that coarsely align the sub-cubes. Those matchings can then be used to align the intermediate processes via the here introduced 'local alignment using 2D image registration' algorithm. The algorithm exceeded the expectations in terms of accuracy and runtime, and excluded the problems with distorted 3D shapes. Additionally, it still can be improved by the approaches mentioned in Chapter 4.1.1, i.e. including probabilities for matches, combining the information of multiple planes, avoiding somata and using a more sophisticated registration algorithm. By introducing these features approximately 90% of the process could be matched reliable and fast. The probabilities can be used to determine which matches are true matches. The remaining 10% can be split into two classes. Firstly, the cases where the overlap of the two sub-cubes is so small that no information can be extracted and secondly, the cases that have enough overlap but still are not matched correctly or not with a high probability. The first class must be detected automatically and then resolved manually. The second will be processed in the next steps of the pipeline. In these steps, a combination of neighborhood knowledge and features is probably the way to success. For example, one could think of using the PointNet autoencoder to extract rich features and additionally use neighborhood information and the already solved agglomerates to match the remaining ones. The already solved cases can also be used to define the size of the overlap and extract the same proportion of agglomerate from both sub-cubes. The automated extraction of features from 3D models can also be improved. Firstly, extracting the 3D models of one process from the overlap has to be solved. The agglomeration can be done with the agglomeration pipeline (Berning et al., personal communication) but needs some work hours of human annotation and parameter tweaking. So far, the agglomeration of the dataset used in this thesis was of low quality due to many merger errors and the dataset had partially very small overlaps between sub-cubes. The small overlap makes it hard to extract the same 'amount' of process from both sub-cubes. This is why one needs to know the overlap which could either be calculated by image registration or by already matched processes. The next part that can be improved is the sampling of the 3D model and the resulting point cloud. Describing a homogeneous surface requires much fewer points than an inhomogeneous one. Hence, one could vary the density of the points within one cell. Smooth structures are sampled sparsely and complicated structures are sampled densely. This allows one to minimize the total number of points while maximizing the contained information. The last reasonable improvement is a rework of the encoding architecture of the PointNet autoencoder. Qi et al. published 2017 an improved version of PointNet, PointNet++. PointNet++ captures local structures induced by the underlying metric space of points which enables the network to learn local features with increasing contextual context (Qi et al, 2017). This is done by hierarchically applying PointNet. Using this improvement might enhance the autoencoder to extract 'better' features. The PointNet autoencoder could also be used for other approaches in the field of connectomics. One obvious application area is the type classification of agglomerates, e.g. if an agglomerate is a dendrite, an axon or a soma. Currently, the classification is done on hand-crafted features which could be complemented by learned features.
Although the local alignment project still has many open tasks, this thesis provided a better understanding of the problems as well as several new tools that can help solving them.

Bibliography

[1] P. Flourens, “Recherches expérimentales sur les propriétés et les fonctions du système nerveux dans les animaux vertébrés”, Crevot (Paris), 1824

[2] M. Helmstaedter, ‘Cellular-Resolution Connectomics: Challenges of Dense Neural Circuit Reconstruction”, Nature Methods, 2013

[3] S. Finger, “Origins of neuroscience : a history of explorations into brain function”, Oxford University Press, 1994

[4] JL. Morgan, JW. Lichtman, “Why not connectomics?”, Nat Methods, 2013

[5] M. Berning, KM. Boergens, M. Helmstaedter, “SegEM: Efficient Image Analysis for High-Resolution Connectomics”, Neuron, 2015

[6] W. Denk, H. Horstmann, “Serial Block-Face Scanning Electron Microscopy to Reconstruct Three-Dimensional Tissue Nanostructure”, Plos Biology, 2004

[7] A. Bogner, PH. Jouneau, G. Thollet, D. Basset, C. Gauthier, “A history of scanning electron microscopy developments: Towards “wet-STEM” imaging”, Micron, 2007

[8] ML. Watson, “Staining of Tissue Sections for Electron Microscopy with Heavy Metals”, Rockefeller University Press, 1958

[9] RM. Haralick, LG. Shapiro, “Image segmentation techniques”, Computer Vision, Graphics, and Image Processing, 1985

[10] L. Vincent, P. Soille, “Watersheds in digital spaces: an efficient algorithm based on immersion simulations”, Transactions on Pattern Analysis and Machine Intelligence, P. 583-598, 1991

[11] GE. Hinton, RR. Salakhutdinov, “Reducing the Dimensionality of Data with Neural Networks”, Science, 2006

[12] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, PA. Manzagol, “Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion”, Journal of Machine Learning Research, 2010

[13] Y. Wang, Z. Xie, K. Xu, Y. Dou, Y. Lei, “An efficient and effective convolutional auto-encoder extreme learning machine network for 3d feature learning”, Elsevier, 2016

[14] M. Brown, DG. Lowe, “Automatic Panoramic Image Stitching using Invariant Features”, International Journal of Computer Vision, 2007

[15] LA. Teverovskiy, OT. Carmichael, HJ. Aizenstein, N. Lazar, Y. Liu, “Feature-Based vs. Intensity-Based Brain Image Registration: Comprehensive Comparism using Mutual Information”, International Symposium on Biomedical Imaging: From Nano to Macro, 2007

[16] CR. Qi, H. Su, K. Mo, LJ. Guibas, “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”, Conference on Computer Vision and Pattern Recognition (CVPR), 2016

[17] CR. Qi, L. Yi, H. Su, LJ. Guibas, “PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space”, NIPS, 2017

[18] HW. Kuhn, “The Hungarian method for the assignment problem”, Bryn Mour College, 1955

[19] Egerváry, “On combinatorial properties of matrices”, Matematikaies Fizikai Lapok, 1931

[20] A. Frank, “On Kuhn’s Hungarian Method - A tribute from Hungary”, EGRES, 2004

[21] H. Bay, T. Tuytelaars, L. Van Gool, “SURF: Speeded Up Robust Features”, Proceedings of the 9th European Conference on Computer Vision, Springer Verlag, 2006

[22] B. Leng, Y. Liu, K. Yu, X. Zhang, Z. Xiong, “3D object understanding with 3D Convolutional Neural Networks”, Elsevier, 2015

[23] Y. Fang, J. Xie, G. Dai, M Wang, F. Zhu, T. Xu, E. Wong, “3D Deep Shape Descriptor”, CVPR, 2015

[24] J. Xie,G. DaiF. Zhu, E. Wong,Y. Fang, “DeepShape: Deep-Learned Shape Descriptor for 3D Shape Retrieval”, IEEE Trans Pattern Anal Mach Intell, 2016

[25] L. Xu, Li, J. Ren, C. Liu, J. Jia, “Deep Convolutional Neural Network for Image Deconvolution”, NIPS, 2014