Universitetet
i Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study programme/specialisation: Information Technology – Automation and Signal Processing	Spring semester, 2018 Open/~~Confidential~~
Author: Simen Walmestad Tofteberg	..... (signature of author)

Programme coordinator: Professor Kjersti Engan

Supervisor(s): Post-Doc Mahdieh Khanmohammadi

Title of master's thesis:
Synaptic Vesicle Detection in Microscopy Images using Convolutional Neural Network and Compressed Sensing

Credits: 30

Keywords: Deep Learning, Convolutional Neural Network, Compressed Sensing, Signed Distances, Synaptic Vesicles, Sprague-Dawley rats.	Number of pages: 70 + supplemental material/other: 2 + attached file Stavanger,...15.06.2018..... date/year

University of Stavanger

Synaptic Vesicle Detection in Microscopy Images using Convolutional Neural Network and Compressed Sensing

MASTER'S THESIS

Simen Walmestad Tofteberg

June 2018

Under the supervision of Post-Doc Mahdieh Khanmohammadi and Professor Kjersti Engan.

Faculty of Technology and Science
Department of Electrical Engineering and Computer Science
University of Stavanger

Abstract

In response to stressful situations, the body activates its sympathetic nervous system with the sudden release of hormones. This increases the presence of adrenaline and noradrenaline which improves muscle strength and endurance. The response is the ”fight-or-flight” reflex that prepares the body to endure emergency situations. The effect is recognizable in increased blood pressure, heart rate and breathing. In everyday life, stress can be triggered by deadlines at work or experienced by a student about to deliver their thesis. A small amount of stress can be helpful to motivate and endure difficult situations. However, being exposed to stress over an extended period of time can lead to depression and may increase risk of early development of dementia such as Alzheimer's disease.

It is hypothesized that being constantly stressed for a prolonged time can affect the signalling between the neurons of the brain. The signalling is a chemical process that uses chemical neurotransmitters stored inside spherical synaptic vesicles. The signal transportation within synapses involves the physical movement of the vesicles towards the pre-synaptic membrane called the active zone. It has been shown that stress influences the distribution of synaptic vesicles.

The objective of this thesis was to develop an automatic synaptic vesicle detection algorithm for use on microscopy images of rat brains. The proposed system is based on a concept of using a Convolutional Neural Network and Compressed Sensing (CNNCS) to predict the synaptic vesicle centers. The neural network is trained to approximate compressed signed distance arrays from different observation axes uniformly distributed around the input image. The approximated compressed signal can be decoded and reconstructed into predicted synaptic vesicle positions in the original image.

Based on the experiments and results, the framework of the system has proven to be robust. However, the different neural networks that has been evaluated has not been able to predict these encoded signals significantly accurate due to lack of diversity in the dataset. The best neural network has a mean squared error of 1.61×10^{-2} which needs to be reduced to 1×10^{-4} in order for the system to be able to predict accurate synaptic vesicle positions.

Preface

This thesis marks the end of my 5-year Master's Degree in Automation and Signal Processing at the Department of Electrical Engineering and Computer Science at the University of Stavanger (UiS).

I would like thank my head supervisor Mahdieh Khanmohammadi for her excellent guidance and valuable inputs through the whole semester.

I would also like to thank Prof. Kjersti Engan for her valuable feedback and advice through the semester.

I want to thank my family for support, patience and encouragement through my whole Master's Degree.

And lastly, I want to show my gratitude towards Aud and Ivar Fett who welcomed me so well to Stavanger five years ago.

Table of Contents

List of Tables

List of Figures

Abbreviations

Symbol:	Definition:
A + PL	Autoencoder and processing layers
Conv	Convolutional
CNN	Convolutional Neural Network
CNNCS	Convolutional Neural Network with Compressed Sensing
FC	Fully-connected
SSH	Secure Shell

Chapter 1

Introduction

1.1 Motivation

In response to stressful situations, the body activates its sympathetic nervous system with the sudden release of hormones. This increases the presence of adrenaline and noradrenaline which improves muscle strength and endurance. The response is the ”fight-or-flight” reflex that prepares the body to endure emergency situations. The effect is recognizable in increased blood pressure, heart rate and breathing. In everyday life, stress can be triggered by deadlines at work or experienced by a student about to deliver their thesis. A small amount of stress can be helpful to motivate and endure difficult situations. However, being exposed to stress over an extended period of time can lead to depression and may increase risk of early development of dementia such as Alzheimer's disease[2].

It is hypothesized that being constantly stressed for a prolonged time can affect the signalling between the neurons of the brain. The signalling is a chemical process that uses chemical neurotransmitters stored inside spherical synaptic vesicles. The signal transportation within synapses involves the physical movement of the vesicles towards the pre-synaptic membrane called the active zone. It has been shown that stress influences the distribution of synaptic vesicles[3].

Rats are observed to respond with similar behaviour to stress as humans. They also possess similar nervous system as humans and is therefore good candidates for conducting such experiments.

1.2 Thesis Objective

In order to be able to study the distribution of synaptic vesicles in the brains, the first step is to detect the position of the synaptic vesicles. This work is usually for a neuroscientist to do, and is an expensive and time consuming process. Therefore, an automatic synaptic vesicle detection

method has been requested.

This thesis focuses on developing an automatic synaptic vesicle detection method for microscopy ssTEM images of rat brains.

1.3 Related Work

In 2017, a method using a small CNN to model synapses in anisotropic images was published with the article "*Toward Streaming Synapse Detection with Compositional ConvNets*"[4]. Raw electron microscopy (EM) images are fed into small parallel CNNs that has each been trained to recognise specific elements of synapses. In the paper, three parallel marginal CNNs looking for neuron membrane, intercellular cleft and synaptic vesicles are combined into a compositional CNN. The method does not discriminate each synaptic vesicle but produces asymmetric densities of vesicles.

Laptev, Veznehnevets, Dwivedi and Buhmann uses a combination of the SIFT Flow algorithm and a random forest classifier to segment ssTEM images in "*Anisotropic ssTEM Image Segmentation Using Dense Correspondence across Sections*"[5]. The SIFT Flow algorithm is used for feature extraction through the volume of the stacked images and a random forest classifier predicts probabilities of the cell membrane positions in each image based on the extracted features. Their best result has a pixel error of 7.9×10^{-2}[5].

An other approach by Dahl and Larsen presents a dictionary learning method for segmenting images[6]. The dictionary is trained on image patches with different textures and labels. Each image patch has a corresponding atom in a label dictionary, and by finding the nearest atom for a image test patch a label patch probability is attached the associated image region. With overlapping patches, several probabilities is attached to each image pixel. The method is shown to be robust to noise, and only experiencing a decline in performance when adding around 15% Gaussian noise[6].

An interesting approach regarding image segmentation is the proposed system from the article "*Fully Convolutional Networks for Semantic Segmentation*"[7]. The developed method is a tuned CNN where the FC layers are transformed into conv layers. Such models has the advantage of processing inputs of arbitrary size because of the system only containing conv layers. The output is trained to approximate ground truth segmented images. The result is pixel-wise predictions for the test images. The method is proved to perform better than popular region-based CNNs (R-CNN) on multiple benchmark segmentation datasets[7].

Xue and Ray introduced a convolutional neural network that combines compressed sensing (CNNCS) to detect cells in microscopy images[8]. The annotated positions in the images are measured from different observation axes uniformly distributed around the images with signed distances. The signed distance arrays are then compressed into an encoded signal with compressed sensing. A CNN is trained to predict the encoded signal, and the output from the model can be reconstructed into image points that corresponds to the cell centers. The proposed CNNCS framework exceeds the accuracy of the state-of-the-art methods on different datasets used

for microscopy cell detection. In comparison to the rat brain images, the size of the cells are much bigger than the vesicles. The vesicles also have closer locations to each other and the contrast in the images are different.

1.4 Proposed Method Overview

Using a CNNCS to predict the synaptic vesicle positions is the proposed solution to the synaptic vesicle detection problem. It has never been used to predict synaptic vesicle positions before, but was the chosen method because of the structure matching the available annotations for synaptic vesicles in the dataset. The method is designed to predict the center positions. Contrary, known existing methods are looking for the synaptic vesicle density regions. In Figure 1.1, an overview of the proposed method is shown.

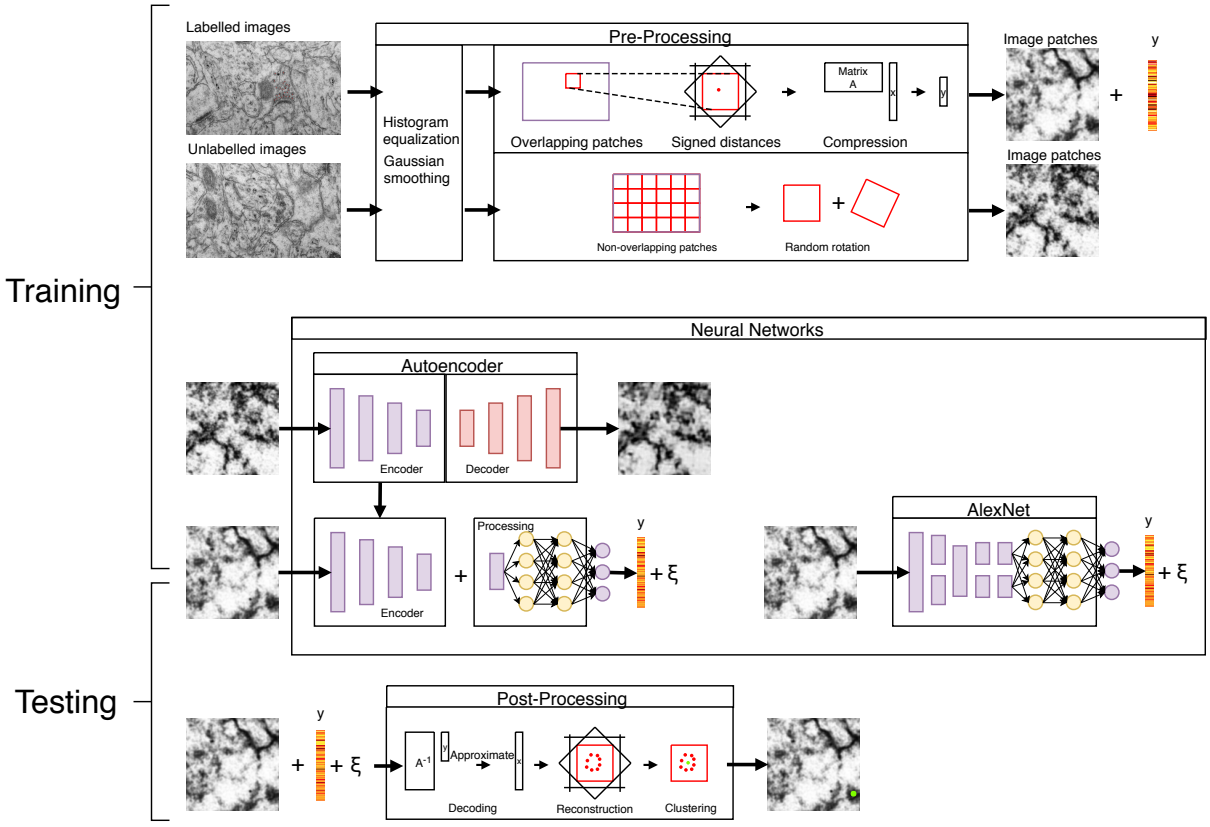


Figure 1.1: Overview of the proposed system. The pre-processed encoded signals, y, is used to train the neural networks to approximate similar signals that can be transformed to image points in the post-processing.

1.5 Thesis Outline

The thesis consists of seven chapters with different sections and subsections plus appendices. The figures used are mainly developed from the dataset and made by the author unless otherwise stated. Below are the different chapters described with a short summary.

- **Chapter 1:** Introduction
  - This chapter describes the thesis outline, objective and related works.
- **Chapter 2:** Background
  - This chapter contains an introduction of the theoretical background for the developed system.
- **Chapter 3:** Data and Materials
  - The chapter represents the original data and materials used in the proposed method and experiments.
- **Chapter 4:** Proposed Method
  - This chapter presents the proposed method with details
- **Chapter 5:** Experiments and Results
  - This chapter contains the conducted experiments and their results.
- **Chapter 6:** Discussion
  - In this part the results from the experiments are discussed with the constructed system.
- **Chapter 7:** Conclusion and Future Work
  - Here the reader is presented with a conclusion on the system performance and suggestions on future work.
- **Appendix:** Program Files
  - This part contains a brief description of the python code used in the thesis.

Chapter 2

Background

This chapter contains the background information on subjects necessary to fully understand the proposed system that will be explained in later chapters. In particular, this includes the theory behind artificial neural networks, signed distances and compressed sensing.

2.1 Biological Neural Networks

A nervous system in a brain consists of intricate patterns of neurons. Neurons are the processing units in the system. Within one cubic millimetre there can be a dense network of more than 10^4 neurons[9, Chapter 1.1]. The density may vary over the sections of the brain, but over all the network of neurons makes the processing of information possible.

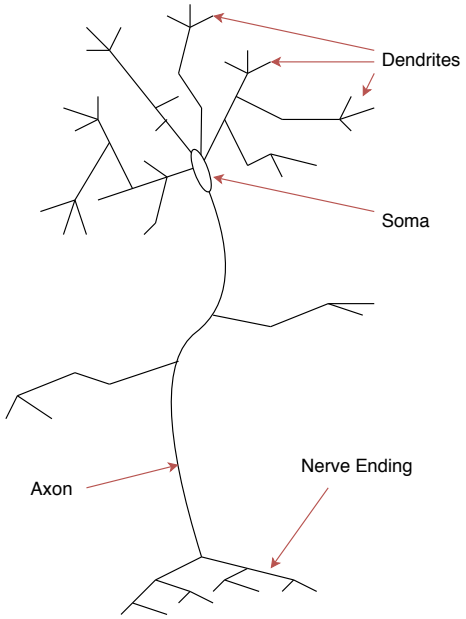


Figure 2.1: Illustration of a biological neuron.

A neuron can be described by three functionally distinct parts as illustrated in Figure 2.1. The dendrites functions as the input receiver. The information gathered from the dendrites are processed inside the soma. The soma activates the axon if the total input arriving is beyond a certain threshold, adding a non-linearity. The axon and nerve ending is then transmitting a signal spike to dendrites of other neurons.

2.1.1 Synaptic Transmission

The neurons interacts through the axon terminals and dendrites. A dendrite connected to an axon terminal forms a synaptic transmission. The pre-synaptic neuron is the cell that is connected with its axon terminal. Upon arrival of the signal spike from the soma, the synaptic vesicles physically moves towards the pre-synaptic membrane and releases the stored neurotransmitters. The post-synaptic neuron captures the neurotransmitters with the post-synaptic receptors. The area between the post- and pre-synaptic neuron is called the synaptic cleft or *active region*. Figure 2.2 provides a brief explanation of a synaptic transmission.

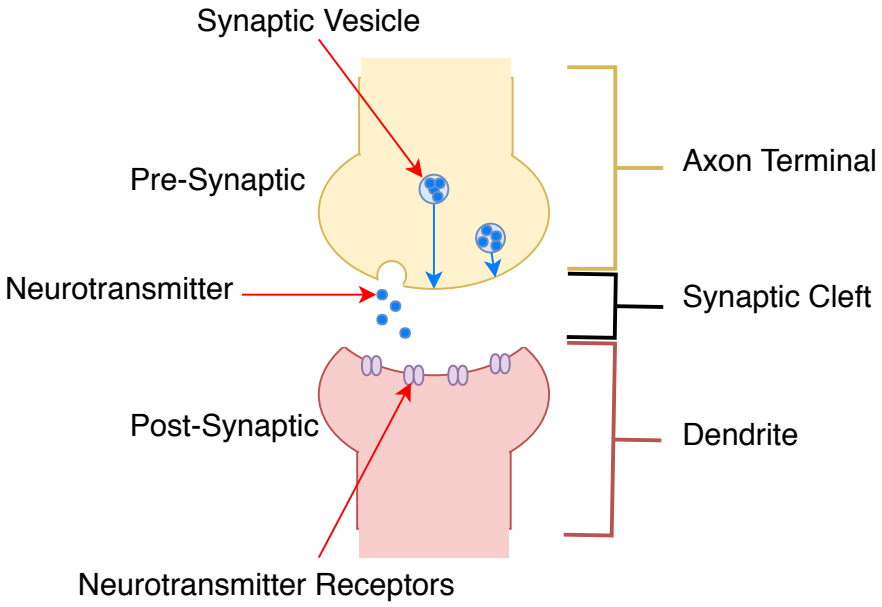


Figure 2.2: Illustration of a synaptic chemical transmission.

2.2 Artificial Neural Networks

The idea of artificial neural networks, that is so prominent in machine learning today, is to mathematically synthesize neural networks that is seen in all advanced biological beings. The large increase in common computer processing power over the past few decades has been the key to its modern success. In particular an artificial neural network consist of three types of layers; input layer, hidden layer and output layer. In order to be described as a deep neural network the system needs to include all of these layers. In Figure 2.3, an artificial neural network containing all of those layers is shown.

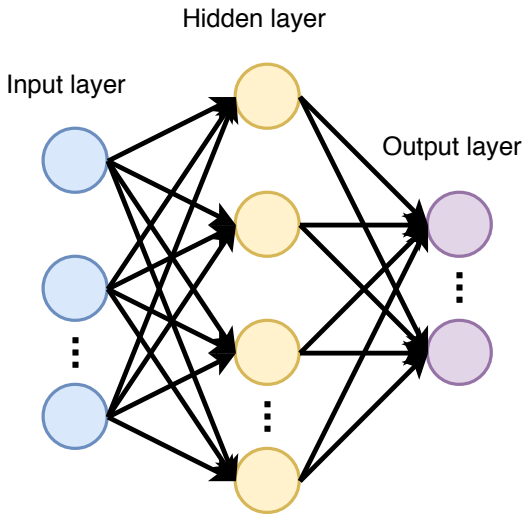


Figure 2.3: An artificial neural net with input layer, hidden layer and output layer.

The input layer is the layer that interacts with the external environment and represents this with a pattern to the rest of the network. The body of the model is constructed by one or more hidden layers that process the input pattern and encode it into the output layer. The output layer process and transmits the information gathered from the hidden layers according to the structure it is given. In a traditional neural network the output corresponds to the predicted label of the inputs. In Figure 2.4, an overview of one artificial neuron is visualized.

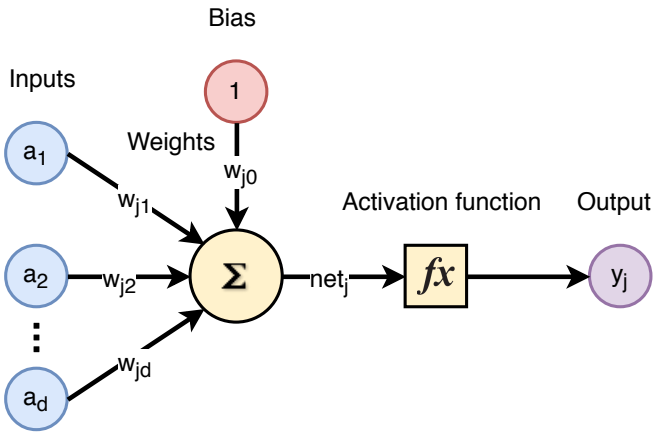


Figure 2.4: An artificial neuron.

Similar to the biological neural networks the artificial neural networks contains artificial neurons or perceptrons. These components are based on the assumption that they mimic the biological process. A neuron at one layer combines the signals from the neurons in the previous layer and uses an activation function before forwarding the signal. All the inputs are multiplied with individual weights, that is to be updated during the training. The weighted inputs are summed in the neuron and in this process it can also be added off-sets or biases that also get updated during training as in Figure 2.4. In Equation 2.1, the net_j is the perceptron j prior to the activation function[10, p. 285].

$$net_j = \sum_{i=1}^{d} a_i w_{ji} + w_{j0} = \sum_{i=0}^{d} a_i w_{ji} \equiv \mathrm{w}_j^T \mathrm{a} \tag{2.1}$$

Where a denotes an array containing d a_i from the previous layer, and w_{ji} denotes the weights and biases for the hidden unit, or perceptron, j. With the activation function as well, the hidden neuron j produces the output, z_j, as in Equation 2.2[10, p. 285].

$$z_j = f(net_j) \tag{2.2}$$

Where the function generates a crucial non-linearity for the outputs. The activation function, $f(\cdot)$, can have different properties and is further discussed in Subsection 2.2.1.

2.2.1 Activation Function

The activation function is the element that ensures non-linearity in the perceptrons. In Equation 2.2, the activation function is the described relation between the output y_j and the perceptron net_j. Another desired property of the activation function is for it to be differentiable for the gradient based optimization methods. Some typical non-linear activation functions includes:

Sigmoid Function

$$f(net_j) = \frac{1}{1 + e^{-net_j}} \tag{2.3}$$

Tanh Function

$$f(net_j) = tanh(net_j) = \frac{e^{net_j} - e^{-net_j}}{e^{net_j} + e^{-net_j}} \tag{2.4}$$

ReLU (Rectified Linear Unit) Function

$$f(net_j) = max(0, net_j) \tag{2.5}$$

2.2.2 Fully-Connected Layer

When a layer in neural network is referred to as a FC layer, it is equivalent to the traditional structure with neurons as presented in section 2.2. Here the neurons of the fully-connected layers have weighted inputs and activation functions as presented in Equations 2.1 and 2.2. The

term fully connected means that the net_j variable of all neurons of a layer is a function of all the outputs from the prvious layer.

2.2.3 Convolutional Layer

A convolutional layer consists of one or multiple filter kernels containing weights as in the FC layers. These are convolved with the input and produces an output that is forwarded to the activation function. Processing images in a neural network is typically performed as 2D-convolution in the conv layers. In Equation 2.6, a general formula for 2D-convolution is expressed.

$$\mathrm{o}(i,j) = \mathrm{a} * \mathrm{h} = \sum_m \sum_n \mathrm{a}(i-m, j-n)\mathrm{h}(m,n) \tag{2.6}$$

Here the 2D-input, a, is convolved with a kernel-filter, h, which produces the output, p. The output dimensions is determined by the *striding* and *padding* in the convolution in addition to the input dimensions. The stride of a convolution is the number of elements the window moves after each computation. The padding extends or cuts the edges in the input in order to obtain an integer size. The padding which extends the inputs with zeros, *zero padding*, is often referred to as *same*-padding, and the technique of removing elements is the *valid*-padding. The output dimension for each convolved dimension can be computed prior to the convolution by the expression in Equation 2.7.

$$\text{output size} = \frac{\text{input size} - \text{kernel size} + 2 \times \text{padding}}{\text{strides}} + 1 \tag{2.7}$$

Since the convolutional layer is spatial, kernel-filter has a smaller dimension than the input, the operation can be seen as doing a feature extraction. One convolutional layer can have multiple kernels, and each kernel together with the activation is said to produce different *feature maps* or *activation maps*. A convolutional neural network (CNN) consits of convolutional layers at the beginning and uses fully-connected layers to further process the features obtained in the feature maps.

2.2.4 Deconvolutional Layer

A deconvolutional operation is the backward process that of a convolutional operation. A deconvolutional layer is often used for up-sampling of feature maps produced from convolutional layers. In order to understand the process behind a deconvolutional layer it has to be studied how a regular convolution can be transformed into a convolution matrix. For example, a kernel of size 2×2 as in the first part of the Expression 2.8, convolved with a input of size 3×3, can be expressed as the convolutional matrix of size 4×9 in the last part of Expression 2.8, times the flattened input of size 9×1.

$$h = \begin{bmatrix} h_1 & h_2 \\ h_3 & h_4 \end{bmatrix} \rightarrow \begin{bmatrix} h_1 & h_2 & 0 & h_3 & h_4 & 0 & 0 & 0 & 0 \\ 0 & h_1 & h_2 & 0 & h_3 & h_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & h_1 & h_2 & 0 & h_3 & h_4 & 0 \\ 0 & 0 & 0 & 0 & h_1 & h_2 & 0 & h_3 & h_4 \end{bmatrix} \tag{2.8}$$

The convolution matrix is a special case of the Toeplitz matrix. Both operations in Equation 2.9 and 2.10 produces the same output, only with different shapes.

$$o = h * a = \begin{bmatrix} h_1 & h_2 \\ h_3 & h_4 \end{bmatrix} * \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} = \begin{bmatrix} o_1 & o_2 \\ o_3 & o_4 \end{bmatrix} \tag{2.9}$$

$$o = Ha = \begin{bmatrix} h_1 & h_2 & 0 & h_3 & h_4 & 0 & 0 & 0 & 0 \\ 0 & h_1 & h_2 & 0 & h_3 & h_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & h_1 & h_2 & 0 & h_3 & h_4 & 0 \\ 0 & 0 & 0 & 0 & h_1 & h_2 & 0 & h_3 & h_4 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \\ a_9 \end{bmatrix} = \begin{bmatrix} o_1 \\ o_2 \\ o_3 \\ o_4 \end{bmatrix} \tag{2.10}$$

In a transpose convolution the objective is to neutralize the convolution operation. The operation in Equation 2.10 is the base for the transposed operator. By transposing the convolution matrix, the input can be estimated by the output, o. The Equation 2.11 shows the transpose operation.

$$\hat{a} = H^T o = \begin{bmatrix} h_1 & 0 & 0 & 0 \\ h_2 & h_1 & 0 & 0 \\ 0 & h_2 & 0 & 0 \\ h_3 & 0 & h_1 & 0 \\ h_4 & h_3 & h_2 & h_1 \\ 0 & h_4 & 0 & h_2 \\ 0 & 0 & h_3 & 0 \\ 0 & 0 & h_4 & h_3 \\ 0 & 0 & 0 & h_4 \end{bmatrix} \begin{bmatrix} o_1 \\ o_2 \\ o_3 \\ o_4 \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \\ a_9 \end{bmatrix} \tag{2.11}$$

The last step is to reshape a back to 3×3. The weights in the transpose matrix does not have to be equal to the ones in the original convolution matrix. There are other methods for reshaping the output of convolution back to the input size. Those other methods includes for instance:

- Nearest neighbour interpolation
- Bilinear interpolation
- Bicubic interpolation

One issue with the deconvolutional layer is that it causes checkerboard artefacts in the produced output image.

2.2.5 Pooling Layer

The pooling layer is a spatial feature extractor. A pooling layer can have various properties. It can be a *max pooling*, *average pooling* or a ℓ_2*-norm pooling* layer. Of those techniques, the max pooling is the most common and is also typically superior in image processing[11]. The operation is used to downsample the input and to prevent overfitting. In Figure 2.5, the max pooling is demonstrated with strides of 2.

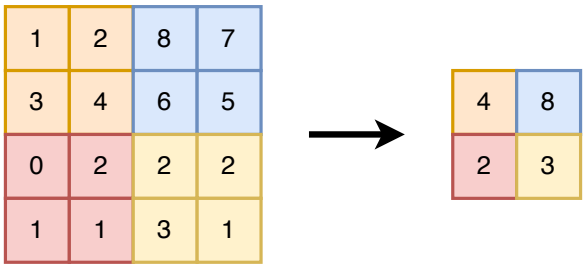


Figure 2.5: Max pooling with strides of 2. Left is the input matrix and right is the produced downsampling with only the most dominant value from each block.

The input matrix of 4×4 is reduced to the output matrix of size 2×2. Only the biggest value is kept from each block.

2.2.6 Autoencoder

An autoencoder is a special case of a neural network where the goal is to be able to reconstruct the input as an output[12, p. 499 - 523]. Although the problem may seem easy, it is often not the output that is the desired element. The model consists of two parts; an encoding part and a decoding part. The encoding part reduces the amount of data, but learns to keep the most valuable information, such that the decoding part is able to approximate the input. In Figure 2.6, the structure of an autoencoder is illustrated.

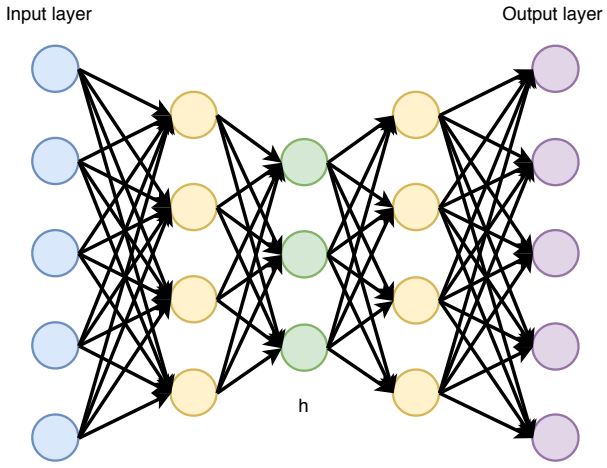


Figure 2.6: An autoencoder with an encoder and a decoder part.

The encoder part can be described as an encoding function as in Equation 2.12[12, p. 499].

$$h = u(a) \tag{2.12}$$

Where a is the input and u the hidden layer that describes a. The compressed data in the hidden layer h, green part of Figure 2.6, has the most important property of data reduction from the input. The decoding section is then reconstructing the signal as in Equation 2.13[12, p. 499 - 500].

$$\hat{a} = g(h) \tag{2.13}$$

Where $\hat{a}$ is the reconstructed a by the compact information stored in the hidden layer h.

For training of an autoencoder the goal is to minimize the difference between the input and the reconstructed output. On the contrary, it is not perfect reconstruction that is desired, but an approximation that stores the key features in the hidden layer h[12, p. 499 - 500].

2.2.7 Loss Functions

In order to evaluate the weights and biases in the neural network after each iteration; a loss function is used. A loss function measures the performance of the output against the desired output or label. The result is then used in a back-propagated *cost function* to update the weights in the network. In Figure 2.7, outputs from a neural network is compared to labels.

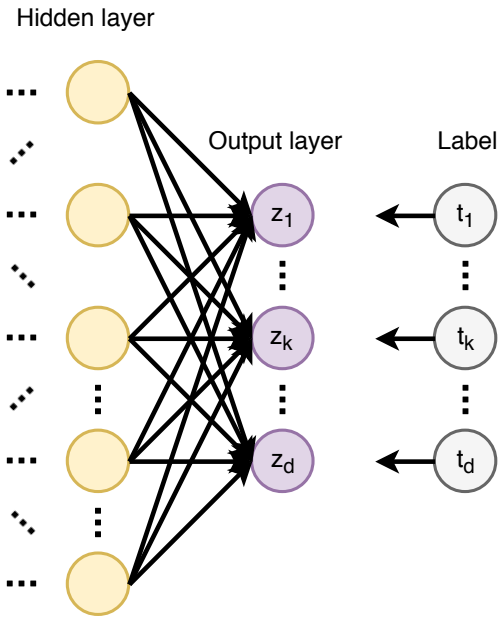


Figure 2.7: A neural network is evaluated on how similar the output is to the label

The loss function is described in Equation 2.14.

$$\text{loss} = L(t_k, z_k) \tag{2.14}$$

Where L is the loss function with the target value or desired output t_k and the produced output from the network z_k for element k in the in the label and output. The loss function can have different properties depending on the output types. For probability outputs and classification structures, the cross-entropy and Hinge loss function are common loss functions. For regression problems it is more common to implement squared error or absolute error. Squared error or ℓ_2-norm loss is calculated as in Equation 2.15.

$$L(t_k, z_k) = (t_k - z_k)^2 \tag{2.15}$$

Where N is the total length of the output and targeted value. The absolute error or ℓ_1-norm loss is formulated as in Equation 2.16.

$$L(t_k, z_k) = |t_k - z_k| \tag{2.16}$$

Since the ℓ_2-norm loss squares the difference, it is more sensitive to big output differences caused by dataset outliers. Therefore, the ℓ_1-norm is more robust except if the outliers is important for the system. In image restoring, the ℓ_1-norm loss can be shown to yeld better results on uniform images[13]. However, ℓ_2-norm loss is generally always used in image processing[13].

2.2.8 Back-Propagation

In supervised learning the goal is to update the networks weights and biases to bring the output of the presented input closer to the desired value. There are different methods of doing back-propagation. The simplest method is *Stochastic Back-Propagation*, where one *epoch* corresponds to the training data being presented once and the weights being updated for each[10, p. 294]. In *Batch Back-Propagation* however, the weights are only updated after each epoch[10, p. 294 - 295]. This is usually the best training method, but since it takes more time to converge it is more feasible with smaller batches when the data is big. Therefore, *Mini-Batch Back-Propagation* is more suitable. The batch is here divided into smaller mini-batches, and the weights are updated after each mini-batch is completed. This allows more updates during one epoch, but not as many as Stochastic Back-Propagation. During Mini-Batch Back-Propagation the *mini-batch loss* or *cost* is calculated after each mini-batch as in Equation 2.17.

$$J(\mathrm{w}) = \frac{1}{bd} \sum_{k}^{bd} L(t_k, z_k) \tag{2.17}$$

Where d is the dimension of the output and where b is the mini-batch size. The cost function can either be the sum or average of the loss. The averaging is more robust for noisy systems. The learning rule of back-propagation is based on gradient descents. The weights, w, is typically initialized with random values or through Xavier initializing which is more optimized based on the activation functions for each layer, such that the variance of the output is equal to that of the weights[14]. Afterwards, the weights are updated in the direction that reduces the cost function. This is pointed out in Equation 2.18[10, p. 290].

$$\triangle \mathrm{w} = -\eta \frac{\partial J}{\partial \mathrm{w}} \tag{2.18}$$

Where η is the learning rate and $\frac{\partial J}{\partial \mathbf{w}}$ is the gradient or partial derivative of J with respect to $\mathbf{w}$. The weights are therefore iteratively updated as in Equation 2.19[10, p. 291].

$$\mathrm{w}(m+1) = \mathrm{w}(m) + \triangle \mathrm{w} \tag{2.19}$$

This is the updating algorithm if all the weights are equally the cause of the loss. In other words; if the model only consists of two layers. If a hidden layer is present as well, the sensitivity of each neuron to the cost is introduced with the chain rule. This leads to Equation 2.20 for hidden layer to output layer weights[10, p. 291].

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = -\delta_k \frac{\partial net_k}{\partial w_{kj}} \tag{2.20}$$

Where the sensitivity of the neuron k is given as δ_k. The sensitivity describes how much the overall loss changes with activation of the neuron[10, p. 291]. The updating of the input to hidden also depends on the chain rule as in Equation 2.21[10, p. 291].

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}} \tag{2.21}$$

2.2.9 Dropout

The term *dropout*, refers to the dropping out of neurons in neural networks. In training, a neuron is present with a probability, p. If the same model is tested, the neurons are always present but the output is weighted with p[15]. In Figure 2.8, dropout in training and testing of a neural network is illustrated respectively.

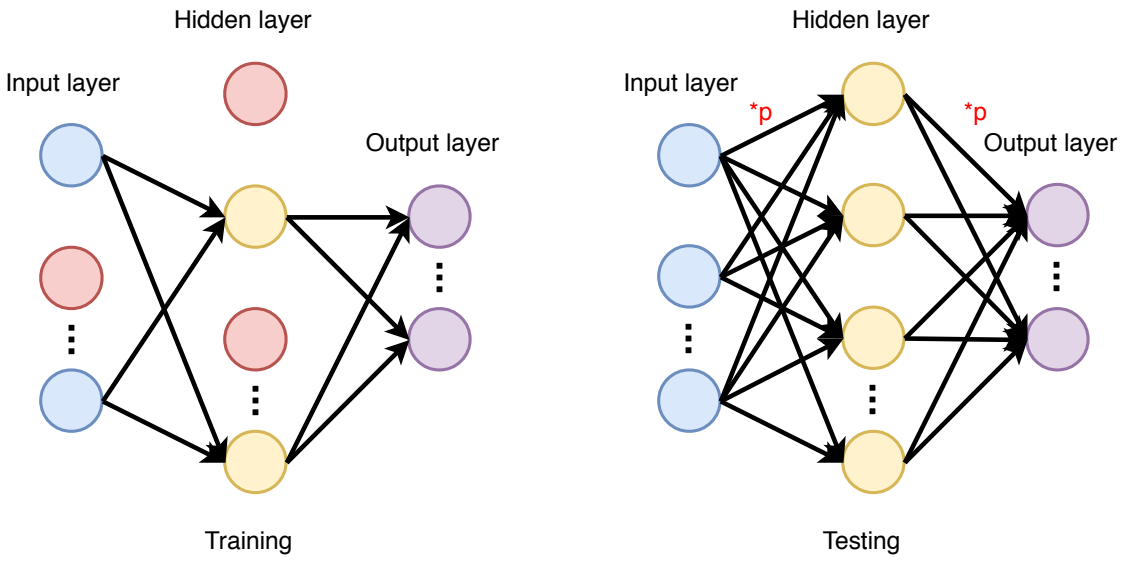


Figure 2.8: Dropout in training of a neural network.

The technique was introduced to prevent overfitting. By randomly removing connections in each iteration the system is not given the chance to settle into an overfitted state. Too much dropout may not be desired either as this can affect the performance. The goal is to find the dropout probability that prevents overfitting, but also is not destroying the model.

2.2.10 Training of Neural Networks

The training of a neural network should be stopped after reaching the global minima. However, there could be reasons for stopping before if it results in overfitting. Overfitting occurs when the performance of the training is improving, but not the validation. A method for preventing this was discussed in Subsection 2.2.9. In Figure 2.9, a training session with validation is shown.

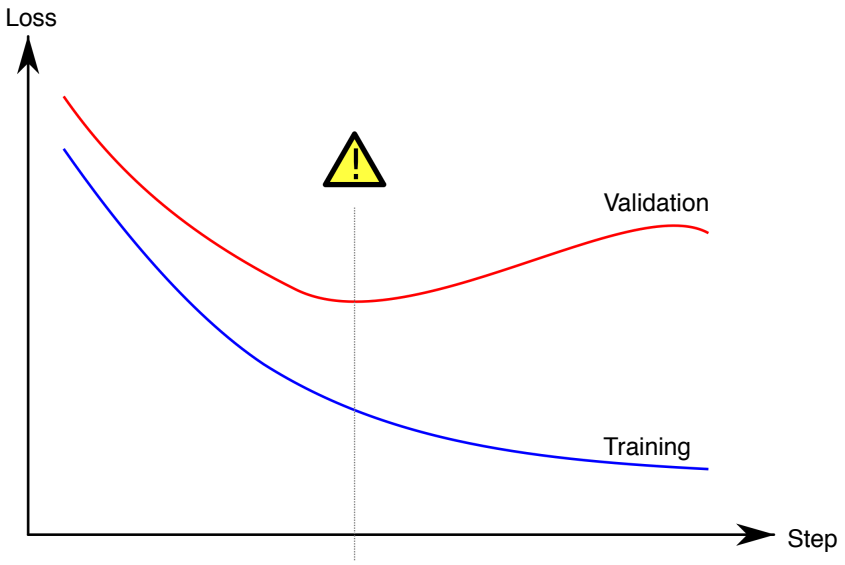


Figure 2.9: Training and validation curves[1].

When the training reaches the area around the yellow sign in Figure 2.9, the validation cost is at its minimum. If the training continues after this, only the training loss improves and the session starts to overfit. To avoid this type of overfitting, the session should be stopped at the yellow sign, or if not already included, dropout could be implemented in a new session.

2.3 Signed Distances

A Signed Distance Function (SDF) is the shortest distance between a closed curve or front and the surrounding points. The distance is signed as positive or negative depending on the point being inside or outside the curve. In Figure 2.10, the signed distances for each pixel to a closed curve is illustrated.

-7	-6	-5	-5	-4	-3	-3	-3	-3	-4	-4	-4	-4	-5	-6	-7
-6	-5	-4	-4	-3	-2	-2	-2	-2	-3	-3	-3	-3	-4	-5	-6
-5	-4	-3	-3	-2	-1	-1	-1	-1	-2	-2	-2	-2	-3	-4	-5
-4	-3	-2	-2	-1	0	0	0	0	-1	-1	-1	-1	-2	-3	-4
-3	-2	-1	-1	0	1	1	1	1	0	0	0	0	-1	-2	-3
-2	-1	0	0	0	1	2	2	2	1	1	1	1	0	-1	-2
-1	0	1	1	1	2	3	3	3	2	2	2	2	1	0	-1
-1	0	1	1	2	3	4	4	4	3	3	3	2	1	0	-1
-2	-1	0	0	1	2	3	4	5	4	3	2	1	0	-1	-2
-1	0	1	1	2	1	2	3	4	4	3	2	2	1	0	-1
-2	-1	0	1	1	0	1	2	3	3	2	1	1	1	0	-1
-3	-2	-1	0	0	0	1	2	2	2	1	0	0	0	-1	-2
-4	-3	-2	-1	-1	-1	0	1	1	1	1	0	-1	-1	-2	-3
-5	-4	-3	-2	-2	-2	-1	0	0	0	0	-1	-2	-2	-3	-4
-6	-5	-4	-3	-3	-3	-2	-1	-1	-1	-1	-2	-3	-3	-4	-5
-7	-6	-5	-4	-4	-4	-3	-2	-2	-2	-2	-3	-4	-4	-5	-6

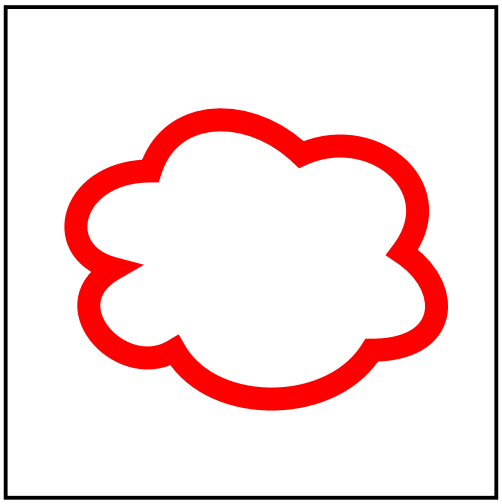


Figure 2.10: Signed distances of an object.

In the right image of Figure 2.10, a closed structure is shown. In the left image, the signed distances for the pixel positions are calculated. The SDF can generally be expressed as in Equation 2.22[16][17].

$$\Gamma(px, py, \Omega) = \begin{cases} d(px, py, \delta\Omega), & \text{if } (px, py) \in \Omega \backslash \delta\Omega \\ 0, & \text{if } (px, py) \in \delta\Omega \\ -d(px, py, \delta\Omega), & \text{if } (px, py) \in R^2 \backslash \Omega \end{cases} \tag{2.22}$$

Where Ω is the front with boundary $\delta\Omega$ that represent the closed structure. The distances can be measured as *city block distances*, as in Figure 2.10 or with Euclidean distances. An *Euclidean distance* can be considered as the hypotenuse length between to points in xy-plane, the city block measures this as the sum of the distance in the x- and y-direction.

In general, SDF is a common technique used in image processing, game developing, geometric computing, reconstruction of surfaces, etc[16].

2.4 Compressed Sensing

Compressed Sensing, or Compressive Sensing (CS) is a compression method that can significally reduce the number of samples in the observed sparse signal and still be able to reconstruct it[18]. By reconstructing through approximation, the method can be able to reconstruct signals from fewer samples than required in the Nyquist Sampling Theorem. The research was not a known field before 2006, but since then has been a key concept for high-dimensional signal compression applied in various areas of mathematics, computer science and electrical engineering[19].

2.4.1 Encoding

The mathematical concept of CS encoding is to randomly undersample the observed sparse signal. This concept relies on the sparsity of the observed signal and the incoherence of the undersampling sensing matrix A. The compressed signal of an observed signal x, y, can be formulated as Equation 2.23[19].

$$y = Ax \tag{2.23}$$

The compressed signal y has dimension M, the sensing matrix A has dimensions $M \times N$ and the observed signal x has dimension N, where $M << N$. Sometimes the observed signal x is not sparse itself, but this can be obtained by a transformation domain. The sparsity can be obtained through a direct transformation, Ψ. The compressed signal, y, can then be expressed as in Equation 2.24[18].

$$y = A\Psi^{-1}X \tag{2.24}$$

Where X is the transform domain coefficients of x, $X = \Psi x$. The matrix A is the incoherence matrix with size $M \times N$. The signal sparsity K, is the number of non-zero elements of N samples. With $M << N$ and $M > 2K$ the output vector y is the compressed signal of length M.

The matrix Ψ^{-1} is the inverse transform matrix. The transformation can be a Discrete Fourier Transform (DFT), Discrete Cosine Transform (DCT), Wavelet Transform etc[18]. A key for making the reconstruction as accurate as possible is to make sure the coherence between the rows of A and the columns of Ψ is minimized[18].

2.4.2 The Sensing Matrix

The accuracy of the reconstruction relies on the construction of the sensing matrix A. For a desirable result, the incoherence has to satisfy the *restricted isometry property* (RIP), as stated in Subsection 2.4.3.

One approach that satisfies these constraints is by using the Discrete Fourier Transformation (DFT) matrix. The formula for DFT is seen as in Equation 2.25.

$$y[k] = \Sigma_{n=0}^{N-1} x[n] W_N^{kn} \tag{2.25}$$

Where $W_N^{kn} = e^{j\frac{2\pi kn}{N}}$. This can be expressed with DFT coefficients as a matrix of size $N \times N$ as in Equation 2.26.

$$\begin{bmatrix} y[0] \\ y[1] \\ y[2] \\ \vdots \\ y[N-1] \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N^1 & W_N^2 & \cdots & W_N^{(N-1)} \\ 1 & W_N^2 & W_N^4 & \cdots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{(N-1)} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)^2} \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ \vdots \\ x[N-1] \end{bmatrix} \tag{2.26}$$

Under-sampling with the DFT matrix is done by skipping rows uniformly. This can be obtained by using a sampling factor such that $k = kq$ in the DFT coefficient. The dimensions of the matrix is then changed to $M \times N$ where $M = \frac{N}{q}$. The undersample DFT matrix is expressed as in Equation 2.27.

$$\begin{bmatrix} y[0] \\ y[1] \\ y[2] \\ \vdots \\ y[M-1] \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W^q & W^{2q} & \cdots & W^{q(N-1)} \\ 1 & W^{2q} & W^{4q} & \cdots & W^{2q(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W^{(M-1)} & W^{2(M-1)} & \cdots & W^{(M-1)(N-1)} \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ \vdots \\ x[N-1] \end{bmatrix} \tag{2.27}$$

In addition to the undersampled DFT matrix there are some other standard sensing matrices:

- ***Gaussian ensembles***: Let the elements in the sensing matrix be randomly sampled from a zero mean Gaussian distribution[20].
- ***Bernoulli ensembles***: Let the elements in the sensing matrix be randomly sampled from a Bernoulli distribution[20].

2.4.3 Restricted Isometry Property

CS relies on the randomness of the sensing matrix. The restricted isometry property (RIP) controls how well the measurements from any sparse input can be distinguished from other measurements with inputs of the same sparsity. The sensing matrix, A, has to satisfy the condition in Equation 2.28[21].

$$(1 - \delta_k)|||x|||_2^2 \leq |||Ax|||_2^2 \geq (1 - \delta_k)|||x|||_2^2 \tag{2.28}$$

Where the expression should be held for all $x \in \mathbb{R}^N$. The global restricted isometry constant, δ_k should be found in the interval $\big[0, 1\big]$.

2.4.4 Decoding

Although the idea of CS is quite new, some of the reconstruction approaches or decoding algorithms in use today dates back to even the early 1970's[18].

Since the sensing matrix in Equation 2.23 has dimensions $M \times N$, with the condition $M << N$, it is an under-determined system with infinitely many solutions. But since x is known to be sparse, the reconstruction algorithms can often benefit from looking for the sparsest solution.

Convex Optimizations

The sparsity of a signal can be described by using the ℓ_0-norm as in Equation 2.29[18].

$$\|x\|_0 = \lim_{p \to 0} \sum_{i=l}^{N} |x_i|_P = \sum_{i=l, x_i \neq 0}^{N} 1 = K \tag{2.29}$$

If the sparsest solution is the desired x, the recovery algorithm can use the knowledge of y to minimise the ℓ_0-norm as in Equation 2.30.

$$\min_x \|x\|_0 \quad \text{subject to} \quad y = Ax \tag{2.30}$$

But as this is said to be $NP - hard$ because of unavoidable combinatorial search, the ℓ_1-norm is more feasible and the closest norm[19]. Therefore, the minimization changes to the *Basis Pursuit* in Equation 2.31.

$$\min_x \|x\|_1 \quad \text{subject to} \quad y = Ax \tag{2.31}$$

For cases where the system is exposed to noise, a conic constraint as in Equation 2.32 is required[19].

$$\min_x \|x\|_1 \quad \text{subject to} \quad \|Ax - y\|_2^2 \leq \epsilon \tag{2.32}$$

Convex optimization algorithms adapted to CS include the following:

- interior-point methods
- projected gradient methods
- iterative thresholding

Greedy algorithms

Greedy algorithms are also a group of algorithms that can be used for finding the sparsest solution of the reconstruction problem. These algorithms are known to be less computational complex than the ℓ_1 optimization techniques, but they are also usually less precise[18].

The most common greedy algorithm is the Orthogonal Matching Pursuit, which succeeded the Matching Pursuit when introduced[19].

2.5 Clustering

When multiple observation axes is applied for the proposed system, the synapse vesicle positions can be predicted through the densities of the reconstructed image points. The clustering must be able to distinguish close densities.

2.5.1 Mean Shift Clustering

A mean shift procedure shifts the data points into the average of the neighbourhood[22]. It aims to locate the mean of local density groups. The clustering method iterates with a window translating towards the local feature density maximum. The kernel can usually be a Gaussian kernel or a flat kernel that has the λ-ball function as illustrated in Equation 2.33[22].

$$K(x) = \begin{cases} 1, & \text{if } \|x\| \leq \lambda \\ 0, & \text{if } \|x\| \geq \lambda \end{cases} \tag{2.33}$$

Where λ is the bandwidth of the local region and determines the window size. The kernel is used to update the mean of the window as in Equation 2.34[22].

$$m(x) = \frac{\sum_{s \in S} K(s-x)s}{\sum_{s \in S} K(s-x)} \tag{2.34}$$

Where s is the data points located in the finite euclidean dataset S. The updated cluster mean is then given as $m(x)$ and the mean shift is equivalent to $m(x) - x$ for the previous cluster mean x. The process iterates until the translation converges. The algorithm does not have a preset number of clusters, uses multiple searches converging towards the different cluster centers. The bandwidth can be set manually or automatic updated for each set of data points. In Figure 2.11, the method is applied to two nearby feature densities.

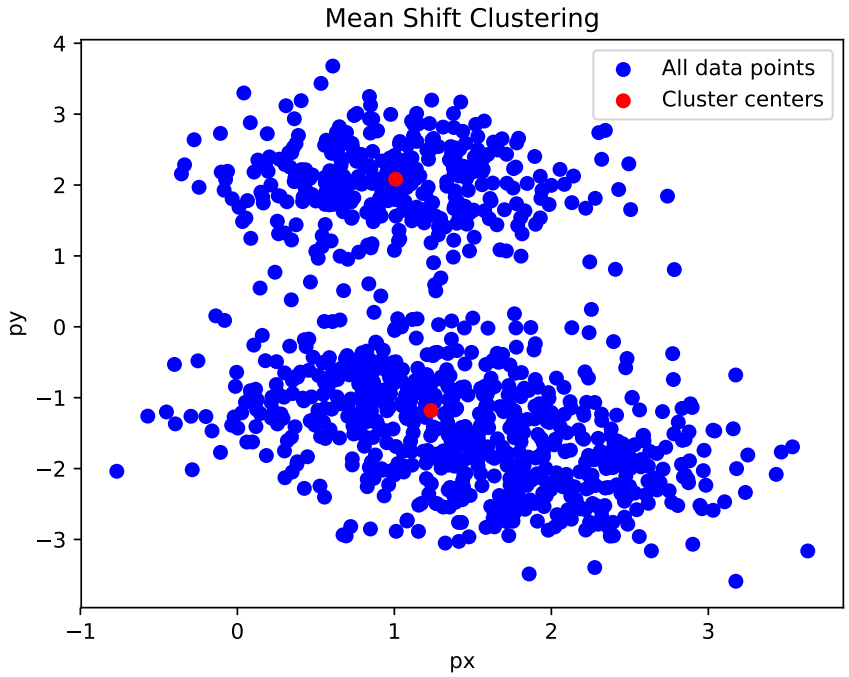


Figure 2.11: Mean shift clustering applied to a dataset of 1000 points.

The blue data points is clustered into two cluster centers, in red, with automatic approximated bandwidth.

Chapter 3

Data and Materials

This chapter contains description of the data and materials that the thesis is based on.

3.1 Animals and Treatment

The data were obtained from male Sprague-Dawley rats (175-200g). These animals were caged pairwise, under a 12 hour day/night cycle. The temperature was constant at 25°C, and the rats were given food and water. Treatment endured for 14 days straight with antidepressant (desipramine, DMI; 10 mg/kg), injected through drinking water. From 5 days before and every 2 days in the experiment the average water consumption was monitored. The procedures involving animals were conducted in accordance with the European Community Council Directive 86/609/EEC and approved by Italian legislation on animal experimentation (Decreto Ministeriale 124/2003-A).

The animals were exposed to to foot-shock stress 24 hours after the last drug injection. The control group were put in the same environment but without the delivered shocks.

3.1.1 Selection of fields of view

From the rat brains six trapezoid-shaped regional samples from dorsal medial prefrontal cortex in both left and right hemisphere were systematically randomly selected. A cross grid was then randomly superimposed over the sections. From the tissue, ultra thin sections were cut with a thickness of 45 ± 5 nm. A Morgagni Transmission Electron Microscope 268 was used to view them. For each block of sections two fields of view were selected. These were first magnified with $11000\times$, and later with $18000\times$. All asymmetric synapses were sampled. In succession, the even and odd numbered synapses went under a final enlargement of $28000\times$. The synapses were followed through the sections of each block and the process stopped when the synapses ceased to be visible.

3.2 Data

In total the experiment was conducted with 24 different rats. From the acquired data, the images in sections has been aligned for statistical analysis purposes. The images also underwent bias field correction. Images from four of the rats has been manually annotated by neuroscientists associated with Aarhus University. In the following experiments conducted in this thesis, the data are divided into to two sets:

- Unlabelled data
  - Containing 20 rats, can be used together with unsupervised methods
- Labelled data
  - Containing 4 rats, consists of the images with annotations

Each rat is identified with NNxx, where xx is the number of the rat. In the original data this identification ranges from NN01 to NN24. In Figure 3.1, an example of an image from the acquired data is shown. The blue ellipse roughly circumference the synaptic vesicles.

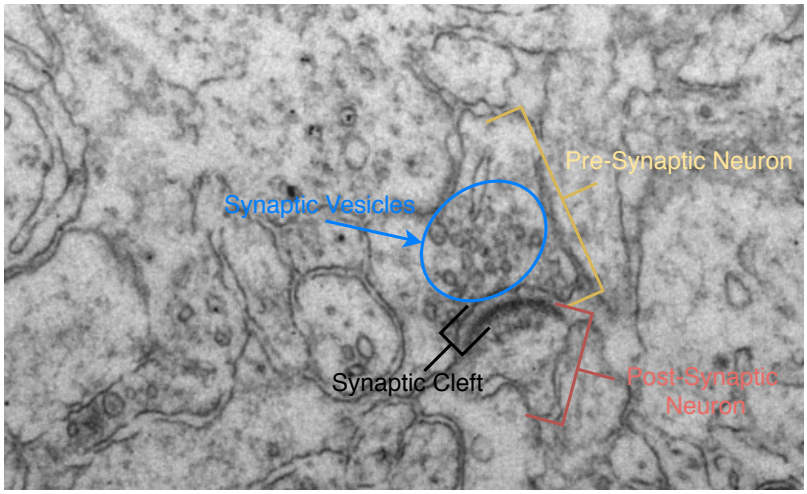
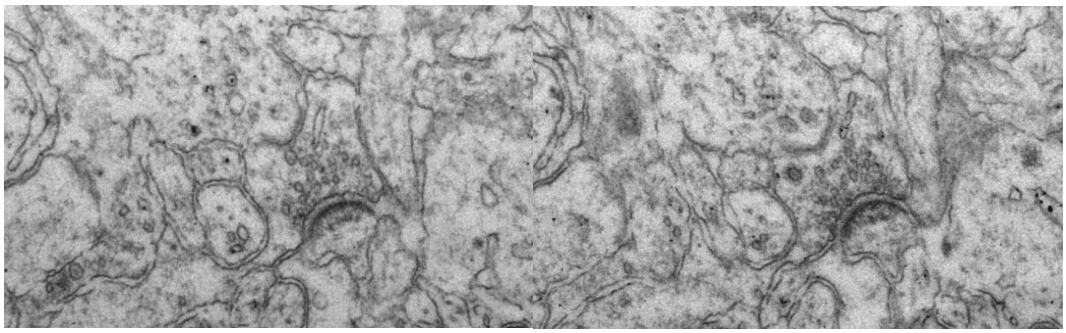


Figure 3.1: An image from the acquired dataset. The vesicles, blue area, are here easily distinguished from other patterns.

Each cut has a thickness of 45 ± 5 nm. Synaptic vesicles in rat hippocampus are studied to have a diameter of 39.0 ± 2.3 nm[23]. That indicates that synaptic vesicles can not be followed through the cuts and that a detection algorithm should be based on 2D images rather than 3D. In Figure 3.2, two consecutive images are shown.



(a) First image. **(b)** Second image.

Figure 3.2: Images from a cylindrical cut with depth interval of 45 ± 5 nm.

From the figure it is verified that the same synaptic vesicles do not appear in two consecutive images. The images look similar despite the new synaptic vesicles and the small changes in the environment. The changes in the synaptic vesicles is equivalent to the changes of other small elements in the cylindrical cut. Hence, the diversity between each cut is minimal.

3.2.1 Labelled Data

The labelled data consists of 68 images from 4 different rats with annotations that point out the center of the vesicles. The images came in three different sizes; 877×533, 1377×1033 and 1177×833. The data origins from 7 different locations for cylindrical cuts into the rat brains. These are annotated as displayed in Figure 3.3.

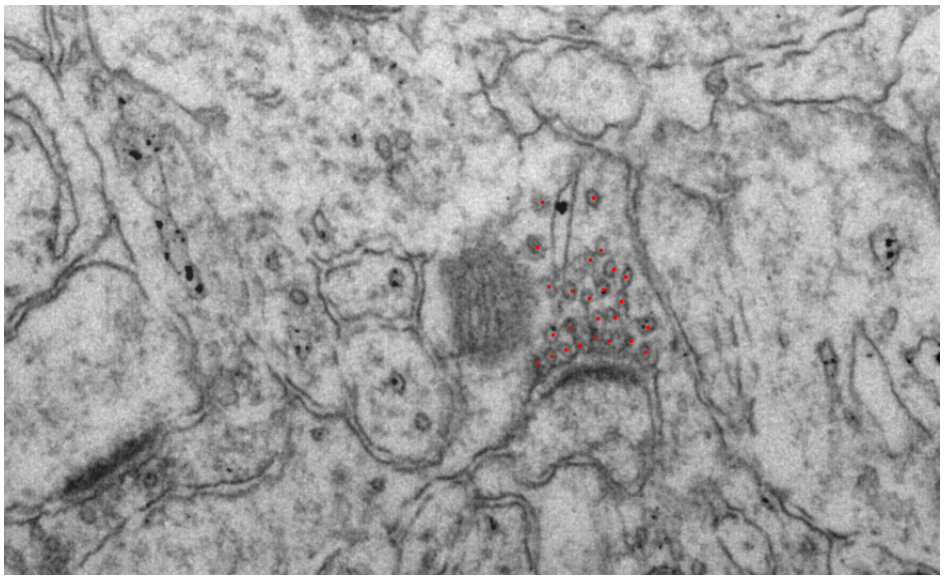


Figure 3.3: The annotated synaptic vesicles(red) from afftransformedimage15 in cut C1 from rat NN11.

In addition to the annotated synaptic vesicles, red points in Figure 3.3, there are several unlabelled vesicles. The annotated synaptic vesicles are the vesicles belonging to visible active

regions. The distribution of images from the different rats and cylindrical cuts can be seen in Table 3.1.

	Rat	Cut	Synapse(s)	Number of images	Containing vesicles
	NN11	C1	9	7	7
	NN11	C4	2, 4 and 6	14	14
	NN11	C4	7	20	17
	NN11	C6	5 and 7	17	11
	NN15	C1	1, 3 and 5	10	9
	NN23	C6	1 and 3	4	4
	NN24	C2	6	11	6
Total:	4	7	13	83	68

Table 3.1: Distribution and origin for the 68 images with annotated vesicles

As seen in Table 3.1, there is only 68 images containing annotations for the vesicles. In order to maximize the dataset potential data augmentation methods are of need.

3.2.2 Bias Field Correction

The original images contains uneven background illumination. The captured images can be shown to have additive bias fields[3]. The annotated images was already processed with bias correction. To make the unlabelled dataset more similar to the labelled data bias correction has been used on this dataset as well with the same method. The bias field was estimated as a quadratic bias field fitted with least squares and removed from the images.

Chapter 4

Proposed Method

This chapter introduces the proposed system and describes the different modules in detail. First pre-processing with image processing, data augmentation, signed distance transformation and signal compression is described. Following this, an introduction of the neural network structures are presented. Before the post-processing is explained.

4.1 System Overview

The proposed system is designed to predict positions of synaptic vesicles in rat brains. The system consists of three main steps: pre-processing, neural networks and post-processing. The pre-processing part extracts image patches from the labelled and unlabelled datasets and encodeds the vesicle positions into encoded signals that is used for training of the neural networks. The neural networks are mainly trained to match the pre-processed encoded signals. In addition, an autoencoder is trained on the unlabelled image patches to perform feature extraction. In the post-processing part the predicted encoded signals is decoded and reconstructed into image positions. If the system uses multiple observation axes, the reconstructed image points are clustered into predicted synaptic vesicle center positions. In Figure 4.1, an overview of the proposed system is given.

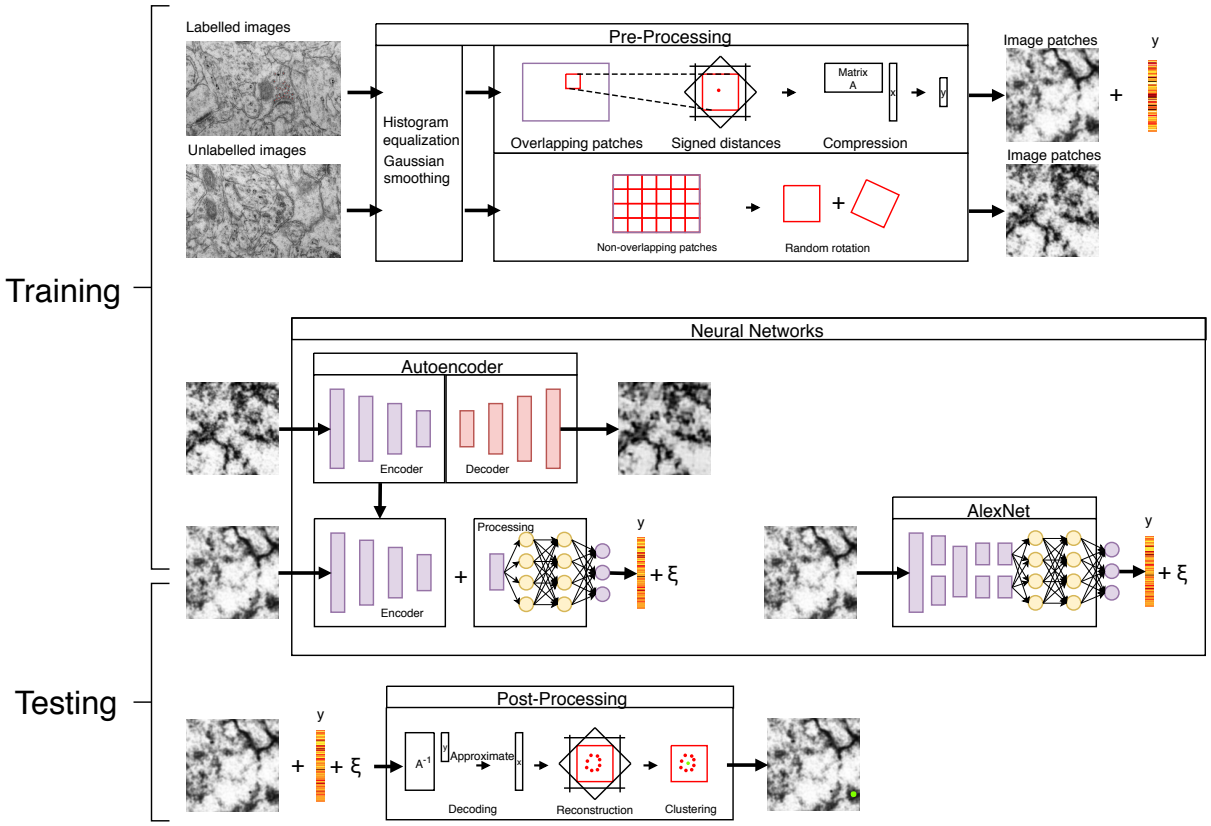


Figure 4.1: Overview of the proposed system including pre-processing, neural networks and post-processing.

The pre-processed encoded signals, y, is used to train the neural networks to produce similar signals for all input images. The trained model is then tested with image patches and outputs a prediction of the encoded signal. Since the model is approximating the encoded signals the results are expected to contain approximation error, ξ. The aim of the training is to reduce this approximation error.

4.2 Pre-Processing

The original images from the unlabelled and labelled datasets are pre-processed to highlight the distinctness between elements in the images through histogram equalization. The images contains noise and is therefore smoothed with a Gaussian kernel. Data augmentation is used to maximize the number of training samples for the neural networks. And the encoded signals are produced from the labelled dataset together with image patches. In Figure 4.2, a schema of the pre-processing is shown.

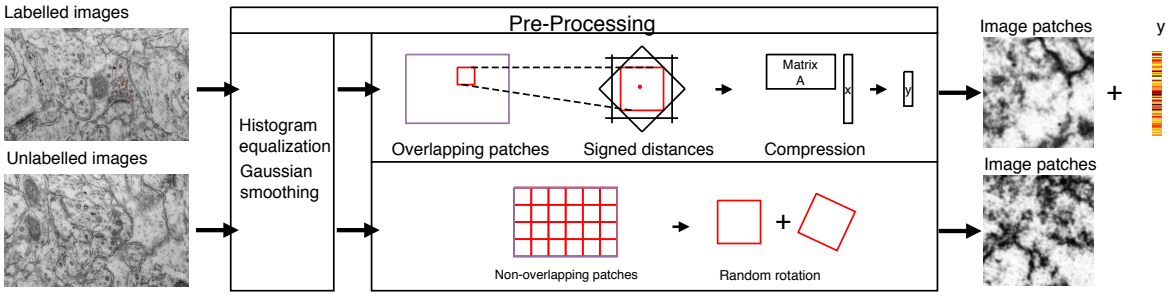


Figure 4.2: Overview of the pre-processing.

4.2.1 Histogram Equalization

In order to increase the feature distinctness in the images, *histogram equalization* was applied to the images. This process distributes the image intensities over the full specter of the image, such that each intensity has the same probability. In Figure 4.3, the histogram equalization of an image is shown.

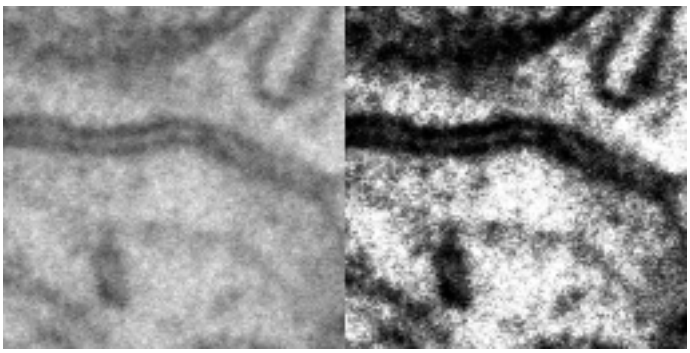


Figure 4.3: Histogram equalization of a brain tissue image.

In the image to the right in Figure 4.3 histogram equalization has been applied. The contrast has increased and the structures in the image has become more distinct. In Figure 4.4, the corresponding histograms to the images in Figure 4.3 are shown respectively.

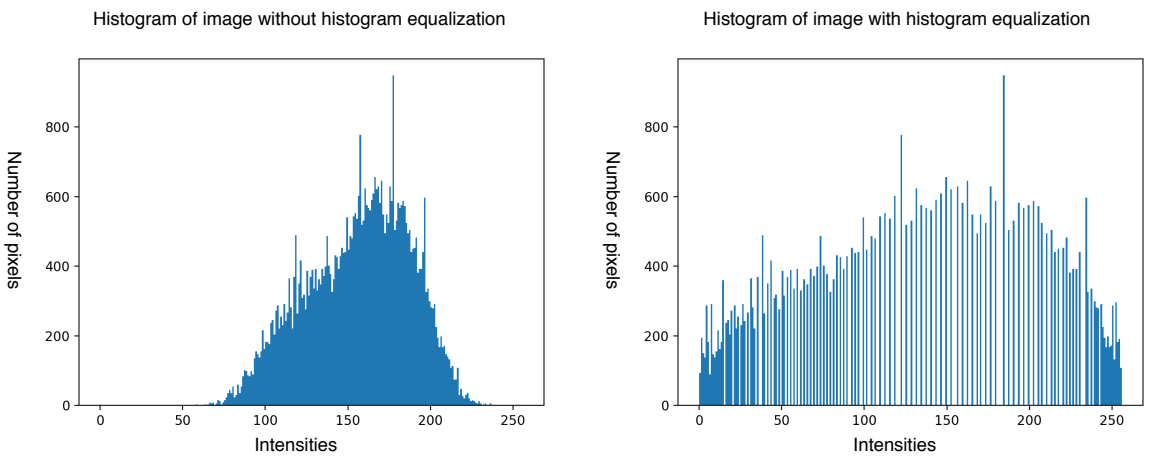


Figure 4.4: Histogram of image before and after histogram equalization

It is clear that the histogram to the right has been distributed over a much wider area than the original distribution in the image to the left. The new distribution have equal probability over small patches throughout the 256 different image intensities.

4.2.2 Gaussian Smoothing

The images was further pre-processed with Gaussian smoothing. Instead of doing a mean filtering where every point inside kernel function is equally weighted, the Gaussian smoothing weighs the points as in a Gaussian distribution. In Equation 4.1, the Gaussian function for a 2D distribution is expressed.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4.1}$$

Parameters for the construction of the Gaussian kernel involves kernel size and standard deviation. In the smoothing of the data, the kernel size was chosen as 3×3. The rows of Pascal's triangle was used to approximate the elements in the kernel. Pascal's triangle with three rows can be visualized as in Figure 4.5.

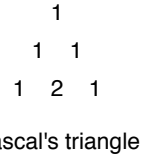


Figure 4.5: Pascal's triangle with 3 rows.

The corresponding Gaussian kernel can then expressed as in Equation 4.2.

$$k = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{4.2}$$

From the $N \times N$ Gaussian kernel the standard deviation can be calculated as in Equation 4.3.

$$\sigma = \frac{2^{N-1}}{\binom{N-1}{\frac{N-1}{2}}\sqrt{2\pi}} = \frac{4}{\binom{2}{1}\sqrt{2\pi}} \approx 0.8 \tag{4.3}$$

The images where smoothed with a 3×3 Gaussian kernel with a standard deviation of 0.8 as shown in Figure 4.6.

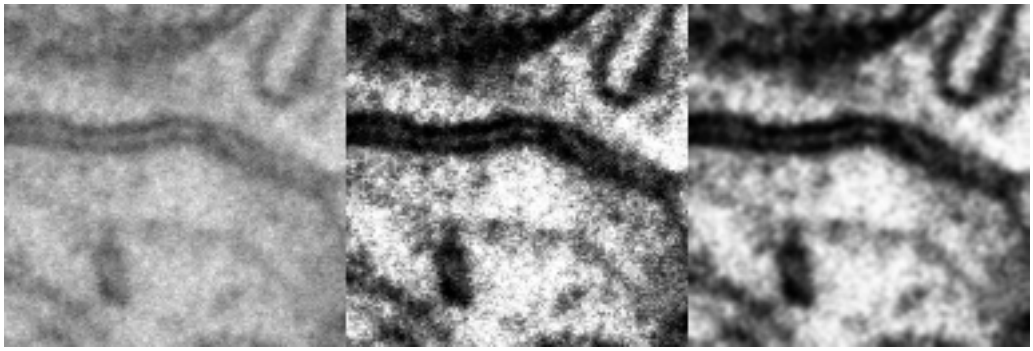


Figure 4.6: Histogram equalization and smoothing applied onto the image to the left.

In Figure 4.6, the image (left) is histogram equalized (middle) and smoothed (right).

4.2.3 Data Augmentation

Data augmentation is technique used to prevent overfitting caused by lack of training data. Figure 4.7 demonstrates how overlapping patches is extracted from the original image (center) with mirror padding.

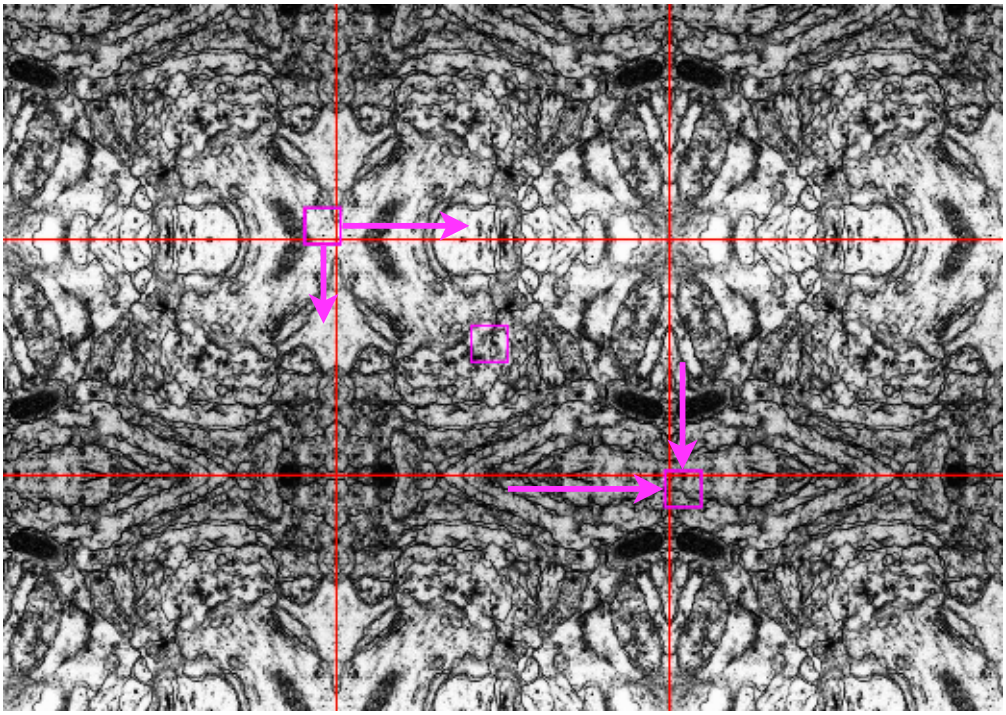


Figure 4.7: Overlapping patches with mirror padding.

The image is firstly mirrored in the horisontal-, vertical- and both axes together with the annotation data. A window of a set size is then translating from the top left corner of the center image to the bottom right corner as shown with purple arrows in Figure 4.7. The windows containing vesicles from the image or padding is extracted as image patches for an augmented dataset. The size of the translation step decides the amount of overlapping and the amount of generated data patches.

Making patches of 128×128 that overlaps with 112 pixels in both directions gives the number of patches as seen in Table 4.1.

	Rat	Cut	Synapse(s)	Patches containing vesicles
	NN11	C1	9	1 465
	NN11	C4	2, 4 and 6	4 691
	NN11	C4	7	8 651
	NN11	C6	5 and 7	2 656
	NN15	C1	1, 3 and 5	2 921
	NN23	C6	1 and 3	1 084
	NN24	C2	6	1 755
Total:	4	7	13	23 223

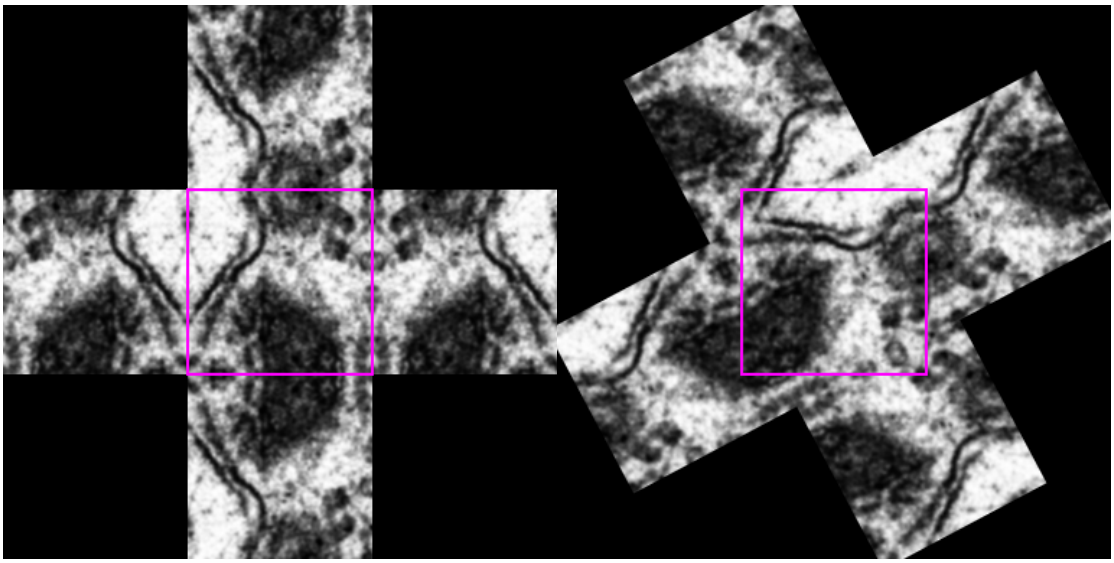
Table 4.1: Image patches with overlapping that contains annotated vesicles.

The dataset containing unlabelled images is augmented with non-overlapping image patches. This is equivalent to putting a grid over each image of the desired size. To match the annotated images, the image patch size was set to 128×128. This resulted in 52065 non-overlapping image patches.

Augmentation by Rotation

The unlabelled images are further augmented with random rotations. The reason for not doing rotational augmentation on the annotated data is because it is later introduced signed distances with multiple observation axes. If the annotated image patch is rotated, it will still produce the same signed distance arrays (if there are a sufficient amount of observation axes).

One random rotation for an unlabelled image patch with mirror padding follows the procedure in Figure 4.8.



(a) Mirror padding without rotation. **(b)** Random rotation.

Figure 4.8: Rotation of an image patch (purple).

The total amount of unlabelled image patches is with random rotations raised to 104130.

4.2.4 Signed Distances

The positions is annotated for the synaptic vesicles. The neural network predicts these through a positional array. For an image of size 128×128 a positional array containing all pixels corresponds to a flattened array of length 16384 for storing positions. To reduce the size of this array, the positions can be transformed by using signed distances from multiple observation axes. The signed distances are produced as illustrated in Figure 4.9.

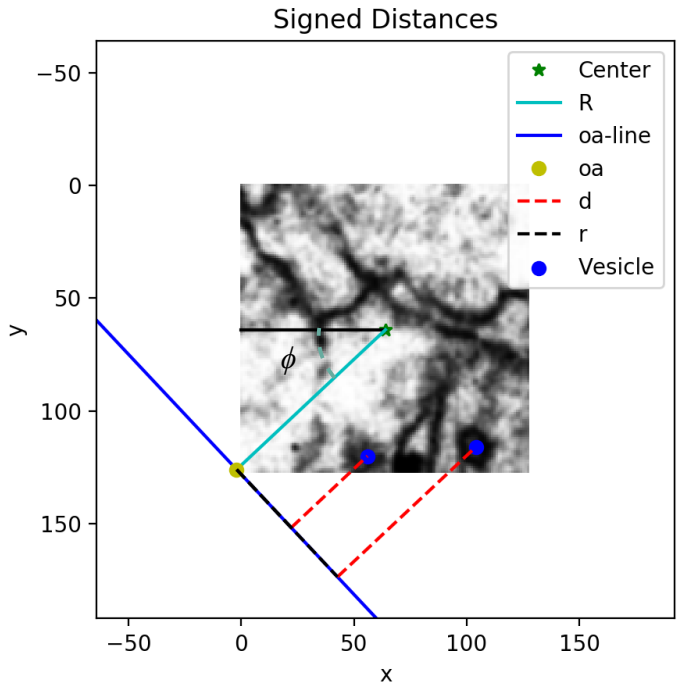


Figure 4.9: Encoding of positions with one observation axis (oa).

The blue line is the observation axis with center in the yellow point. In the lower parts of the image patch, two vesicles can be observed. The distance r corresponds to the offset from the center of the observation axis to the point on the tangent which is placed the closest to each vesicle, illustrated by the black dashed line in Figure 4.9. The signed distance d is the perpendicular distance between the observation axis tangent and the vesicles. In Equation 4.4, the radius r is calculated.

$$r = (py - oa_y)cos\phi + (px - oa_x)sin\phi \tag{4.4}$$

Where px and py is the coordinates for the vesicles, and ϕ is the angle for the observation axis, where the center of the observation axis is denoted oa, as in Figure 4.9. Equation 4.5 is the formula for the signed distance from the tangent of the observation axis.

$$d = -(py - oa_y)sin\phi + (px - oa_x)cos\phi \tag{4.5}$$

With one observation axis the system is vulnerable to noise. To reduce this vulnerability it is introduced multiple observation axis as in Figure 4.10.

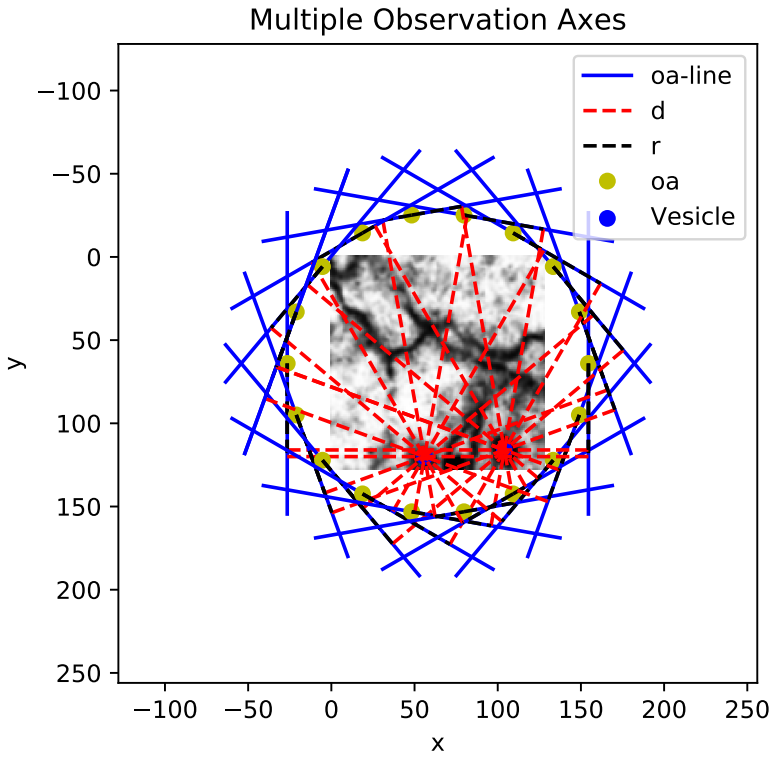


Figure 4.10: Image patch with 18 observation axes.

Each observation axis is placed at a circle around center of the image with a radius of $R = \sqrt{(m/2)^2 + (n/2)^2}$. All the vesicle positions are stored in one array for each observation axis. With an array of length $2R$, one vesicle position is stored as element number $R + r$ with the element being d.

4.2.5 Encoding

In order to further compress the output layer of the neural net, the sparsity of the signed distance array can be exploited. With CS the reduction in size of samples can be greater than the Nyquist Sampling theorem. In Figure 4.11, the signed distance array x is sampled by the underdetermined sampling matrix A.

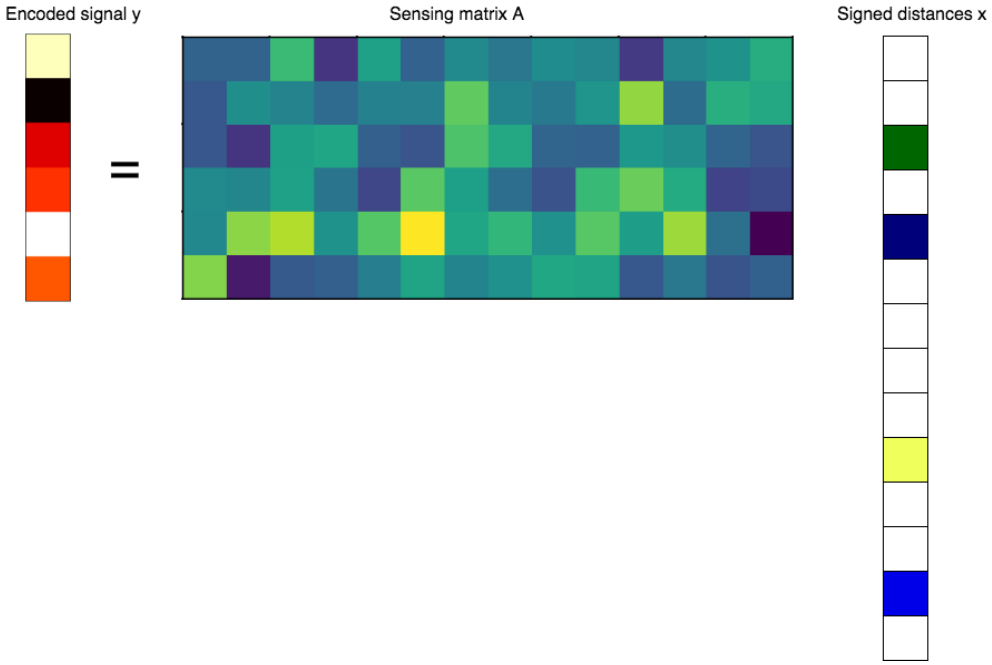


Figure 4.11: Compressed sensing performed on a sparse array.

Here the dot product of the sensing matrix and the signed distance array results in an encoded signal y. The reduction in size is determined by the amount of rows in matrix A. If A has the size $M \times N$, the corresponding encoded signal y is of size M. The incoherence of the sensing matrix plays a big part in how well the decoding algorithm is able to distinguish signals of same sparsity. Therefore, the sensing matrix has to satisfy the restricted isometry property (RIP) from Subsection 2.4.3.

For multiple observation axes the encoding is done separate for each. The resulting encoded signals are then concatenated into one array.

4.3 Neural Networks

The neural networks task is to approximate and predict the encoded signals. The system does not consist of a regular classifier as in typical neural network systems. Instead the output produces as many elements as in the length of the encoded signal. Therefore, the output can be expressed as the encoded signal with approximation error as noise as in Equation 4.6.

$$y_{out} = y + \xi = Ax + \xi \tag{4.6}$$

Where y_{out} is the produced output, y is the ground truth encoded signal and ξ is the approximation error generated by the model.The smaller the approximation error, the better the system can predict the location of the synapse vesicles. To track the progress of the training and updating of the weights it was introduced a cost function of squared error loss as in Equation 2.15. For the system this is implemented as in Equation 4.7.

$$J(\mathrm{w}) = \frac{1}{M} \sum_{k=0}^{M-1} \xi_k^2 \tag{4.7}$$

Where ξ_k is the k^{th} element of the approximation error array of total length as the encoded signal length, M.

The search for finding the most optimal model was based on three main categories:

- Feature extracting with encoder of autoencoder trained on unlabelled dataset, followed with processing layers trained on labelled dataset
- Pre-trained AlexNet-structure with trainable weights and biases
- Pre-trained AlexNet-structure without trainable weights and biases

4.3.1 Autoencoder

To benefit from the amount of unlabelled data, an autoencoder is used to pre-train the first feature extracting or convolving layers of the model. The aim for the autoencoder is to reduce the size of the input images but still keep the most important information so it is possible to reconstruct the input. In Figure 4.12, An autoencoder consisting of an encoder and a decoder part is shown.

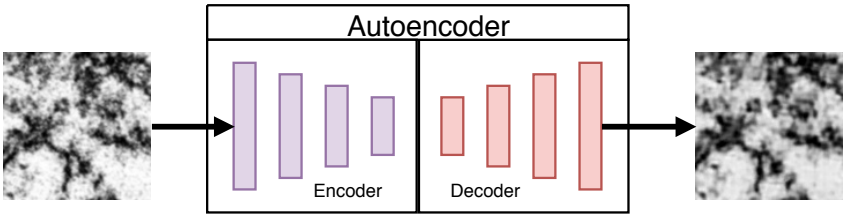


Figure 4.12: Structure of an autoencoder. With an encoder and a decoder part.

Encoding Constraint

Traditional lossy autoencoders compresses the input with a compression percentage beyond 90% in the hidden layer. But as the main goal of the proposed system is to calculate the positions of the synaptic vesicles, it can be beneficial to lower the compression percentage in order to not lose too much valuable features about them. The vesicles are small and traditional autoencoders will benefit from removing their features in order to maintain high compression without losing

too much about the whole image. The compressed encoding output is constrained to have a size of $16 \times 16 \times 16$. This constraint has the compression percentage as in Equation 4.8.

$$\text{Compression} = 1 - \frac{\text{Encoded output size}}{\text{Input size}} = 1 - \frac{16 \times 16 \times 16}{128 \times 128 \times 1} = 0.75 = 75\% \tag{4.8}$$

It was also decided that the autoencoder should pre-train 4 convolutional layers that could be used in addition to added layers for the detection problem.

4.3.2 Processing Layers

After the autoencoder is finished training for feature extraction from the unlabelled dataset, the encoder part is used to produce compact features for more processing. The encoder part is used on the labelled dataset with added processing layers which is then trained to estimate the encoded signals. An overview of this structure is illustrated in Figure 4.13.

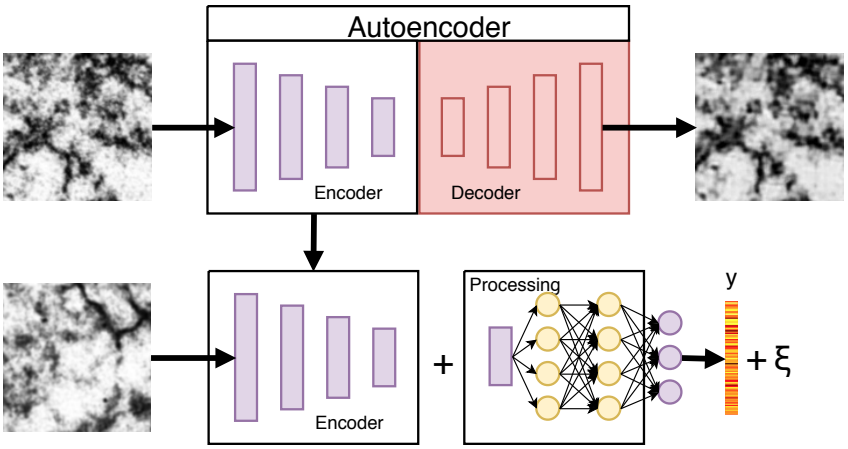


Figure 4.13: Overview of A+PL structure.

In traditional neural networks the output is structured to predict labels of the inputs with probabilities ranging from 0 to 1. To be able to predict the encoded signals the network is tuned into a regressor that outputs continuous values. The number of outputs of the neural network has to match the size of the encoded signal array. Using CS sensing matrices with means around zero and signed distance arrays, the encoded signals can be expected to contain both positive and negative continuous values. Therefore, the activation of the output layer has to be able to produce negative values which is not obtained using functions such as ReLU.

To find the model that predicts these encoded signals the best, different structures and parameters is evaluated to find the network that produces the least cost. For a CNN these parameters corresponds to the list in Table 4.2.

Parameter:	**Description:**
Batch size	Number of samples in each mini-batch
Epochs	Number of times the entire training set is processed
Learning rate	Determines magnitude of the updating direction in gradient descent
Dropout	Probability of leaving a neuron out in training
Number of conv layers	Number of feature extracting layers
Number of filter kernels	Number of produced feature maps from a conv layer
Kernel size	Size of the 2D kernels
Pooling	Type of downsampling
Number of fc layers	Number of processing layers
Number of neurons	Number of units in a fc layer
Activation functions	Type of non-linearity for each layer

Table 4.2: Parameters that can be evaluated in a CNN model.

4.3.3 AlexNet

In the paper that demonstrated CNNCS-models, the pre-trained model of AlexNet was used[8][24]. This model consists of 5 conv layers followed by 3 FC layers. Designed by the SuperVision Group for the 2012 ImageNet competition; where they won with 10.8 percentage points to the runner-up[25]. The competition tasks involved image classification and detection using bounding boxes. The trained weights and biases used in the ImageNet competition is available online[1]. Figure 4.14 displays a brief overview of the AlexNet structure, which is further explained in Table 4.3.

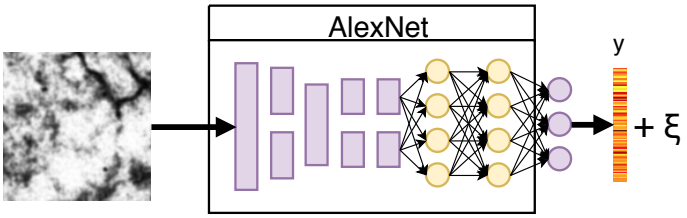


Figure 4.14: Overview of the AlexNet structure.

[1] http://www.cs.toronto.edu/~guerzhoy/tf_alexnet/

Param\Layer:	Conv1:	Conv2:	Conv3:	Conv4:	Conv5:	FC6:	FC7:	FC8:
Groups	1	2	1	2	2			
Kernel size	$[11, 11]$	$[5, 5]$	$[3, 3]$	$[3, 3]$	$[3, 3]$			
Stride	$[4, 4]$							
Nk	96	256	384	384	256			
Pooling	max	max			max			
Pooling size	$[3, 3]$	$[3, 3]$			$[3, 3]$			
Pooling stride	$[2, 2]$	$[2, 2]$			$[2, 2]$			
lrn	used	used						
Af	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	
Nn						4096	4096	os

Table 4.3: Overview of the AlexNet structure. Abbreviations: param = parameters, nk = number of kernels, af = activation function, lrn = local response normalization, nn = number of neurons, os = output size.

The network is trained on colour images and is therefore tuned with a new input layer that fits the monochrome rat brain images, but the attributes from the original structure is kept. Because of the difference in input size, the output from the conv layers vary from the original model, the first FC layer is modified and reinitialized with random weights following the Xavier initializer[14]. The last layer is also modified into the dimensions of the encoded signal. This means that the tuned model has the pre-trained layers from the original AlexNet in layers conv2, conv3, conv4, conv5 and fc7.

The parameters that can be evaluated for AlexNet is listed in Table 4.4.

Parameter:	Description:
Batch size	Number of samples in each mini-batch
Epochs	Number of times the entire training set is processed
Learning rate	Determines magnitude of the updating direction in gradient descent
pre-trained weights	Trainable or not trainable pre-trained weights

Table 4.4: Parameters that can be evaluated in the AlexNet model.

4.4 Post-Processing

The output from the model described is decoded back to signed distances using CS reconstruction algorithms. Each vesicle should have one estimated position for each observation axis. All reconstructed points is clustered into predicted positions. A brief overview of the post-processing is illustrated in Figure 4.15.

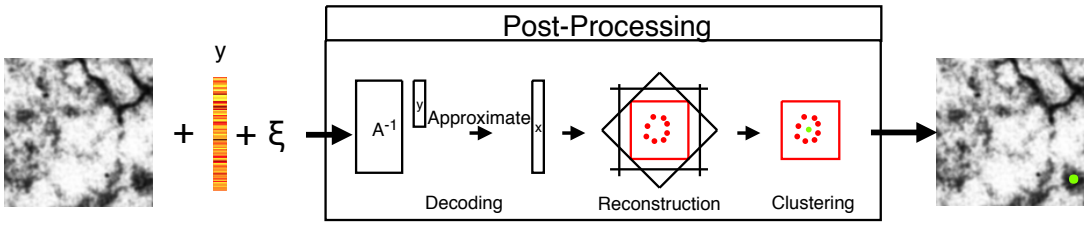


Figure 4.15: Overview of the post-processing.

4.4.1 Decoding of Compressed Signals

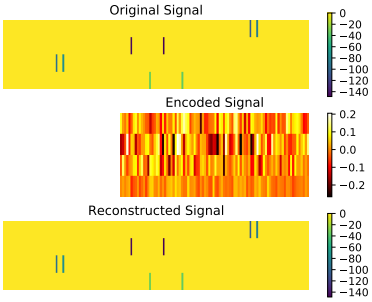
To reconstruct the signed distances from the predicted encoded signals the basis pursuit from Equation 2.31 is used. The equation states that the optimization algorithm should minimize the ℓ_1-norm of the reconstructed array as expressed in Equation 4.9.

$$\min_x \|x\|_1 \quad \text{subject to} \quad y = Ax \tag{4.9}$$

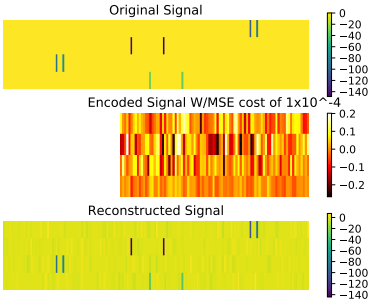
Where the signed distance array, x, is found from the sensing matrix, A, and the encoded signal y.

The optimization problem is solved with a primal-dual interior-point method from the CVXPY package in python called ECOS[26] [27].

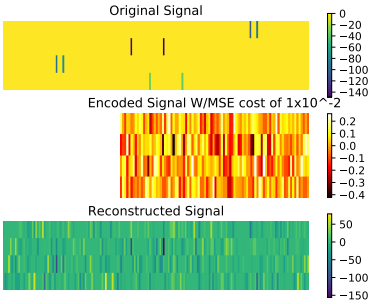
In Figure 4.16, the original sparse signed distances of four observation axes is encoded into four smaller arrays and then reconstructed with and without noise.



(a) No noise.



(b) With noise corresponding to MSE cost of 0.0001.



(c) With noise corresponding to MSE cost of 0.01.

Figure 4.16: Original signal of 4 observation axes compressed and then reconstructed with ℓ_1-minimization.

4.4.2 Reconstruction of Signed Distances

From the decoded signed distances the image positions is reconstructed. Here the input parameters is the reconstructed r and d, which was discussed in Subsection 4.2.4. In addition, the knowledge of the observation axes is also kept from the first transformation. The reconstruction is therefore possible with the Equations 4.10 and 4.11.

$$px = oa_x + rsin\phi + dcos\phi \tag{4.10}$$

$$py = oa_y + rcos\phi - dsin\phi \tag{4.11}$$

Where px and py are the predicted vesicle positions, r and d is the radius and distance shown in Figure 4.9 and ϕ is the angle between the observation axis and the x-axis.

4.4.3 Clustering

To predict the vesicle centers with multiple observation axes, clustering is used to combine the corresponding points from the different axes. In the proposed system this is implemented with

mean shift clustering. It takes the reconstructed image points and bandwidth as input parameters. The bandwidth is the size of the window used for clustering. By automatic calculations for every batch of input points the optimized local bandwidth can be found. This works well if it is more vesicles present than used observation axes. For example if there are only one vesicle and three observation axes this will often result in two or three clusters due to the bandwidth being too low. Instead, the bandwidth were calculated for a dataset exposed to noise and used globally. In Figure 4.17, an image with two close vesicles is shown with mean shift clustering.

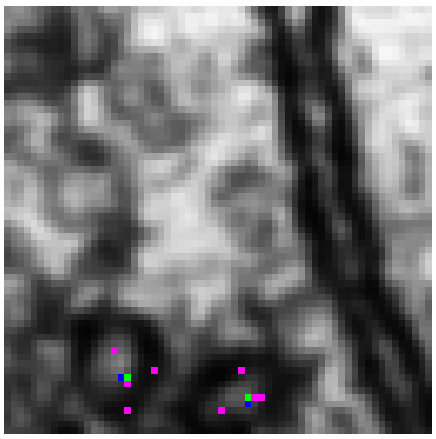


Figure 4.17: Mean shift clustering with a preset bandwidth of 10.0.

In the figure it is used four observation axes with added noise. The blue points are the ground truth vesicle positions, the purple are all the points from the observation axes and the green points are the cluster centers. The bandwidth of 10.0 is able to distinguish the two nearby vesicles.

4.5 Implementation

The proposed system is implemented in a python environment. The data augmentation and training of neural networks has been performed on the Unix servers at the University of Stavanger, due to more computational power. The Gorina 4 server is available for students and consists of three Tesla P100 GPUs. The server is linux based and mainly intended for machine learning purposes. In Figure 4.18, the workflow of the thesis can be shown.

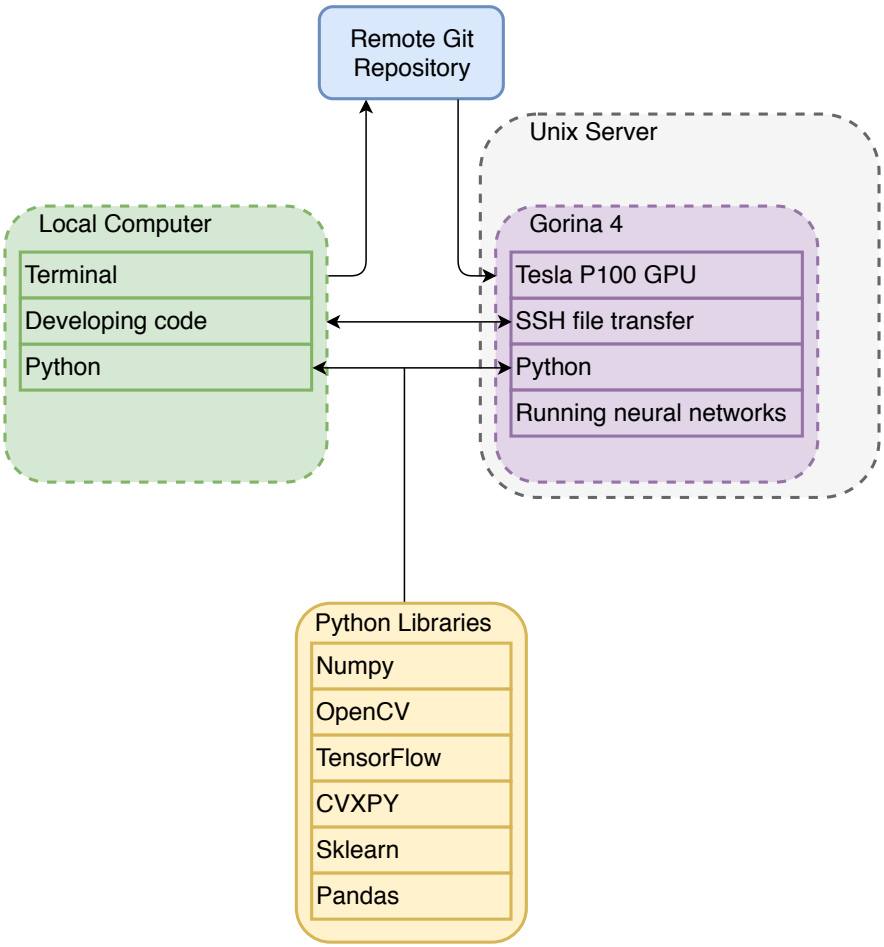


Figure 4.18: Overview of the implemented system.

The python environment on the local computer is cloned into the Unix servers through a remote repository. This lets development of code to be done at the local computer and the running on the Unix server. Results are transferred through secure shell (SSH) back to the local computer. The python libraries is described under:

- Numpy[28]
  - Numpy is the numerical and mathematical backbone of all calculations done
- OpenCV[29]
  - OpenCV is used in the image processing and for reading and writing data
- TensorFlow[30]
  - TensorFlow is the library for all processing and construction of neural networks
- CVXPY[26]
  - CVXPY is the convex optimization library that implements the ECOS solver

- Sklearn[31]
  - Sklearn is the scikit-learn library in python and here used for mean shift clustering
- Pandas[32]
  - Pandas is used for loading the annotations and for storing results from experiments

Chapter 5

Experiments and Results

The chapter ”Experiments and Results”, consists of the experiments and evaluations conducted that verifies the performance of the proposed method. This includes experiments on compression error, robustness to noise, finding the optimal autoencoder structure and evaluations of best neural network structure.

5.1 Compression Error

With compressed sensing it is possible to encode and reconstruct signals accurately if the original signal is sparse and the resctricted isometry property is fulfilled. To test how the compression loss can impact the image positions, 100 signed distance arrays from 100 image patches (one array each) was encoded and reconstructed with different compression rates. The reconstructed signal was then transformed into pixel positions as well as the original signed distance array for ground truth. The error was calculated in euclidean loss. In the experiment the arrays were shown to have a mean k sparsity of 10.1. The testing was done with one Bernoulli sensing matrix with a success probability of 0.5 and a Gaussian sensing matrix with standard deviation of 0.1. The results can be shown in Figure 5.1.

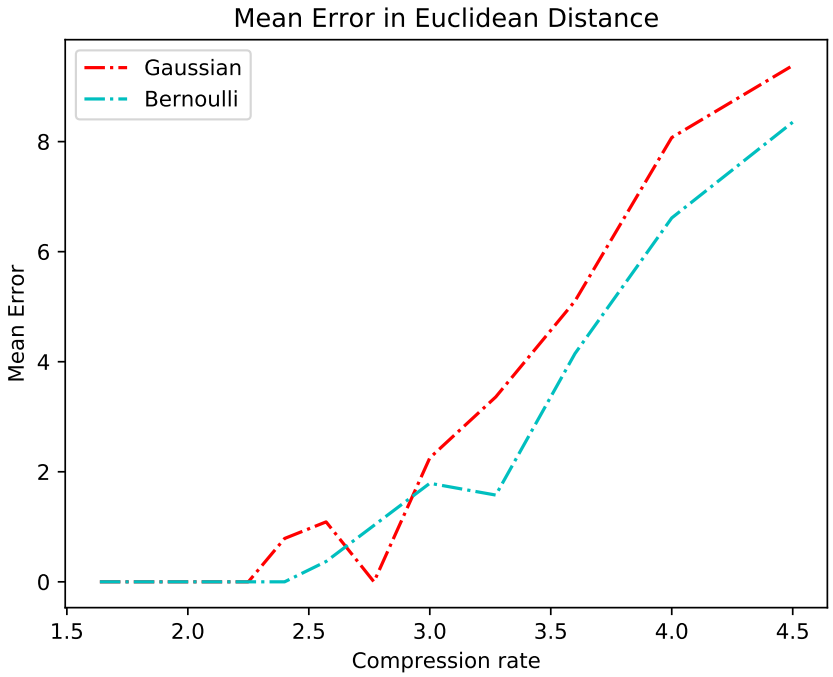


Figure 5.1: Mean Error in image positions for each compression rate.

The result shows that with both ensembles the compression rate can be 2 without impacting the reconstruction. This means that the total amount of labelling data can be halved, which again speeds up the training.

5.2 Robustness to Noise

In the proposed system, a neural network is approximating the compressed signals from the signed distances. This means that the environment is exposed to approximation error. The neural network output can therefore be expressed as in Equation 4.6 and 5.1.

$$y_{out} = Ax + \xi \tag{5.1}$$

Where ξ is the added noise corresponding to the approximation error. The model wants to minimize ξ, but as it is very unlikely that it learns perfect approximation, it is necessary to study how robust the remaining framework of the system is to noise. It was therefore experimented with how Gaussian noise with different standard deviations would impact the system.

The experiment tested hundred images with different number of observation axes. The encoded signals were added Gaussian noise with the standard deviations presented in Table 5.1.

Parameter:	Description:
Images	100
Sensing Matrices	Gaussian, Bernoulli
Observation Axes	$1, 4, 8, 16, 32$
Noise Type	Gaussian
Standard Deviations	$0, 0.0001, 0.001, 0.01, 0.1, 1$

Table 5.1: Parameters used in the experiment.

The signals were then reconstructed into signed distances, and further into pixel-positions before they were clustered to represent the vesicle positions. The precision were measured with the Euclidean distance to the original positions. This was further processed into the mean error for all vesicles in the hundred images. The results can be shown in Figure 5.2.

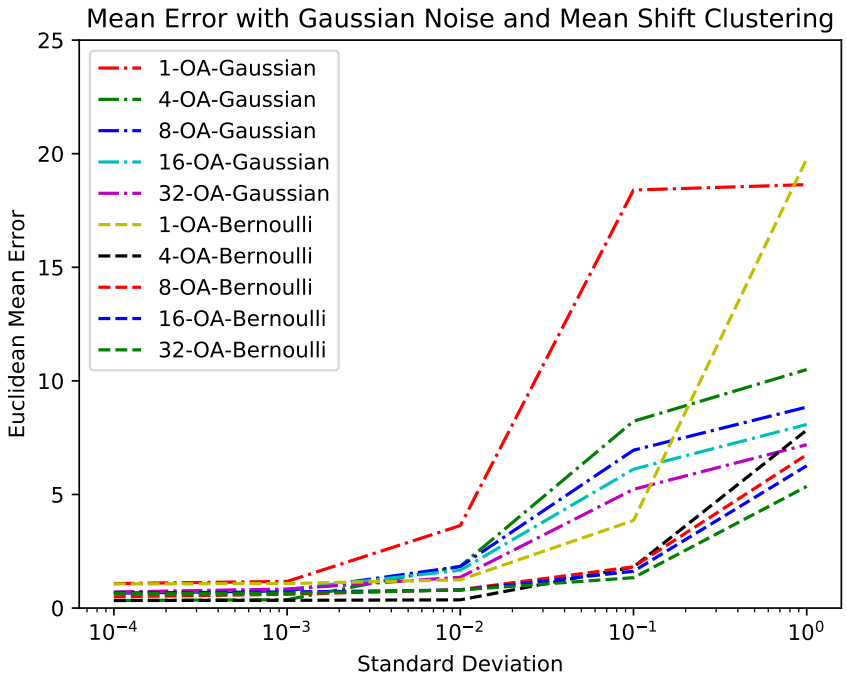
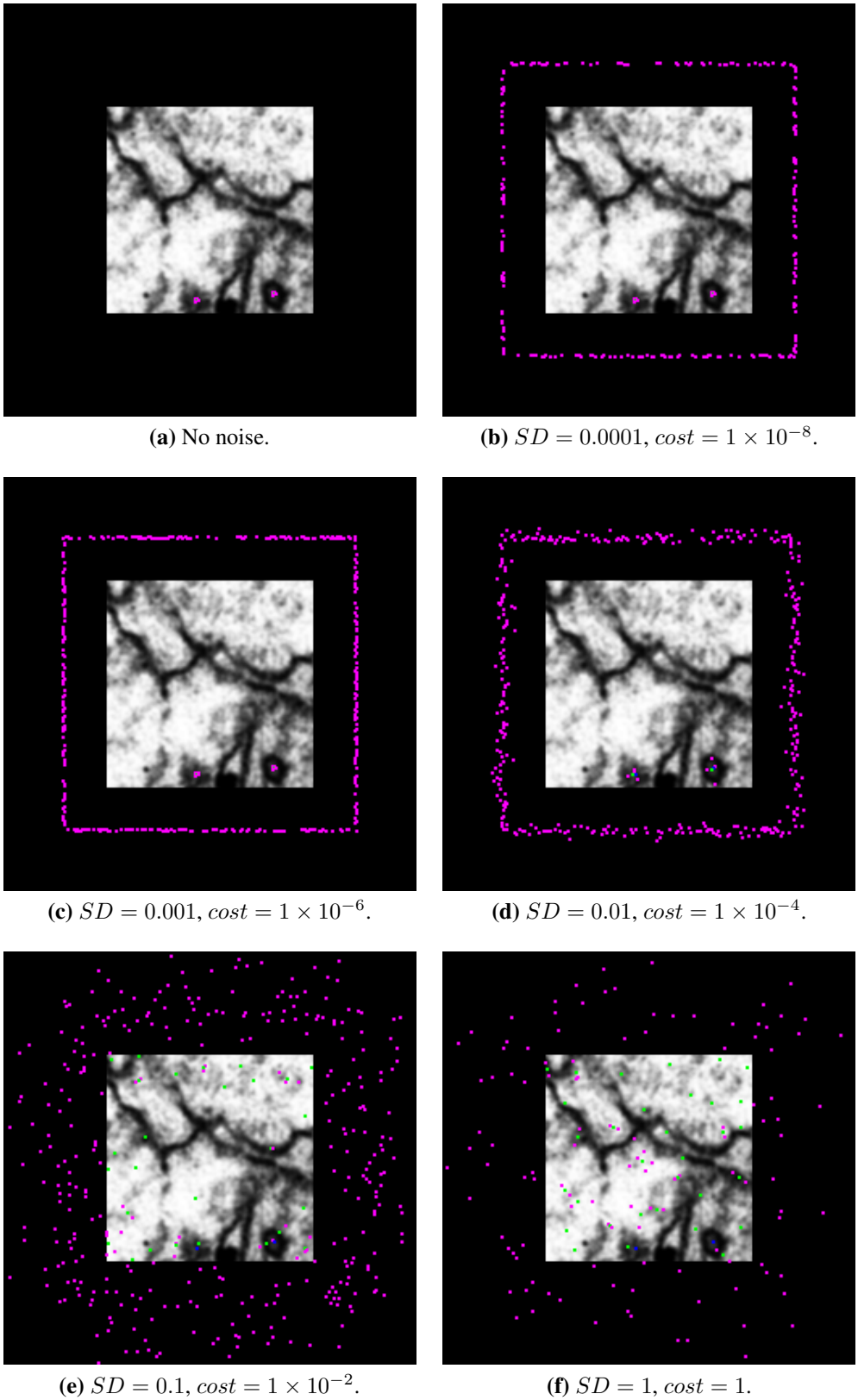


Figure 5.2: Shows the relation between noise and error in the predicted synaptic vesicle positions for 1, 4, 8, 16 and 32 observation axes with Bernoulli and Gaussian ensembles.

As it can be seen from the figure, the more observation axes the more robust the system are to noise. All the different setups misses the vesicles with fever than a mean of 5 pixels with the standard deviation being smaller than 0.01. It can also be seen that the Bernoulli ensemble is slightly better until the noise reaches a standard deviation of 0.1.

The impact of the noise can also be interpreted from the images in Figure 5.3. The reconstructed image points and predicted vesicle centers with and without noise using four observation axes are visualized. The blue dots are the original positions, the green are the cluster centers and the

purple pixels corresponds to all points from the nonzero elements of the reconstructed signed distances.



(a) No noise.

(b) $SD = 0.0001$, $cost = 1 \times 10^{-8}$.

(c) $SD = 0.001$, $cost = 1 \times 10^{-6}$.

(d) $SD = 0.01$, $cost = 1 \times 10^{-4}$.

(e) $SD = 0.1$, $cost = 1 \times 10^{-2}$.

(f) $SD = 1$, $cost = 1$.

Figure 5.3: Visualization of the impact noise has on the system with four observation axes.

5.2.1 Evaluation of Robustness to Cost

In Equation 4.7, the MSE cost of the output from the neural network was the mean of the squared approximation error. Figure 5.4 illustrates how different cost values from the neural network could impact the post-processing.

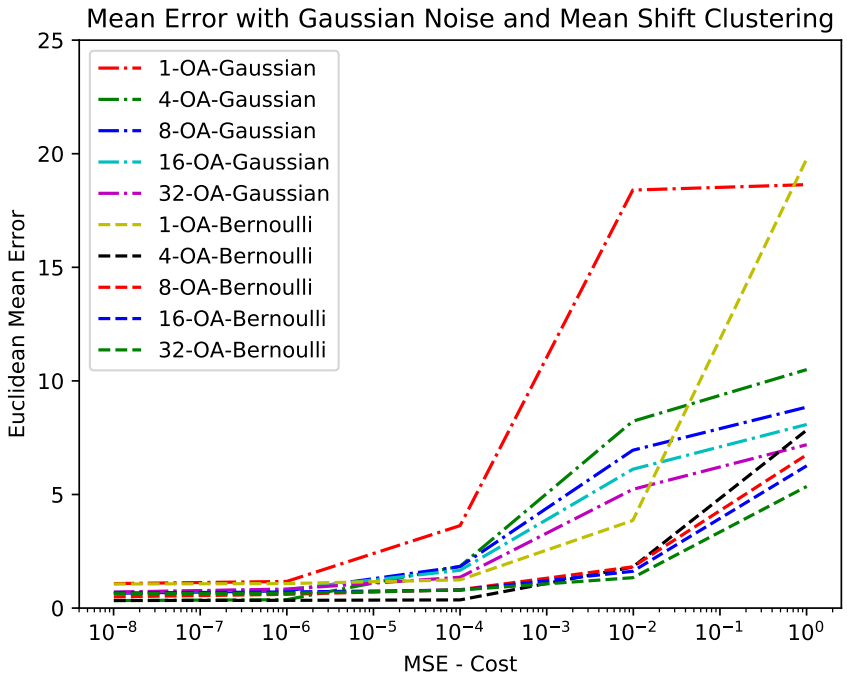


Figure 5.4: Visualization of the relation between cost in the approximated compressed signal and the corresponding image positions.

From Figure 5.4 it is visualized that the system predicts the positions within a radius of 5 pixels with cost up to 1×10^{-4} for all numbers of observation axes and with both Bernoulli and Gaussian matrices. It is also seen that the Bernoulli matrix is more robust to higher cost than the Gaussian.

5.3 Evaluation of Autoencoder

The autoencoder is supposed to train the first feature extracting layers of the model. To minimize the amount of structures that is tested, some parameters of the neural net are kept constant. The structure was set with four convolutional layers in the encoder, including max pooling as pooling and ReLU as activation function for all layers. Max pooling is used because it is expected to perform the best with CNN[11]. Also the mini-batch size is kept at 100 and the input size is 128×128. The input data pipeline is constructed from the unlabelled dataset, which is from different rats than the labelled data. In Table 5.2, the rest of the tested encoder parameters are shown.

Parameter:	Description:
Train set	104130 image patches
Epochs	100
Learning rates	$0.1, 0.01, 0.001, 0.0001$
Filter kernels (Encoder)	$[64, 32, 32, 16]$, $[32, 64, 32, 16]$, $[32, 32, 64, 16]$, $[64, 64, 32, 16]$
Filter size (Encoder)	$[3 \times 3]$
Output size layer $d \times d$	$[64, 64, 32, 16]$, $[64, 32, 32, 16]$, $[64, 32, 16, 16]$

Table 5.2: Parameters used in testing of autoencoder.

The encoder part is mirrored into the decoder part. This means that the decoder starts from the size of $16 \times 16 \times 16$ for all model structures and outputs an image of 128×128. The only significant difference is that deconvolutional layers replaces the convolutional. The models were trained with Adam optimizer and a MSE mini-batch cost function[33].

5.3.1 Verification of Best Autoencoder

The different autoencoders were evaluated on the training curve. The best structure was further trained to 200 epochs. The training curve can be shown in Figure 5.5.

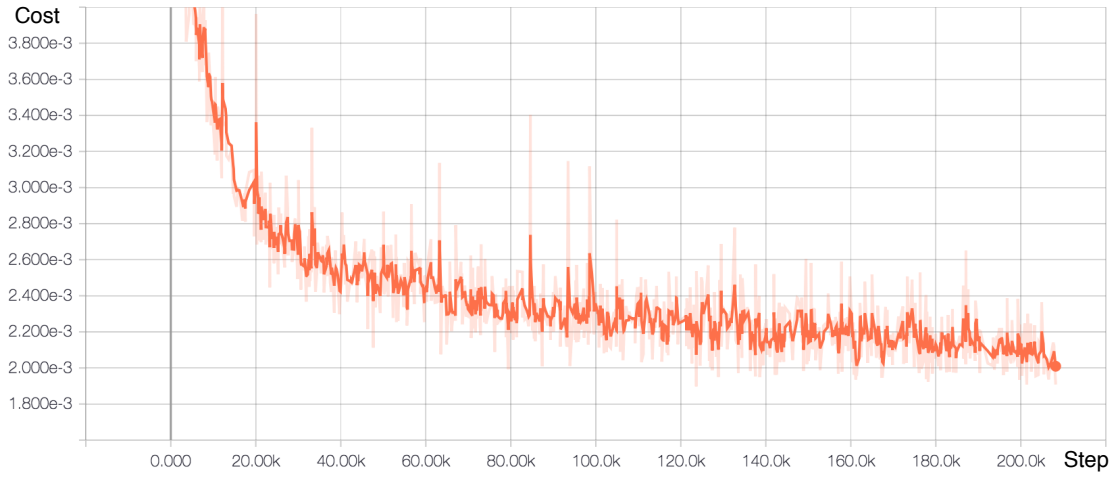
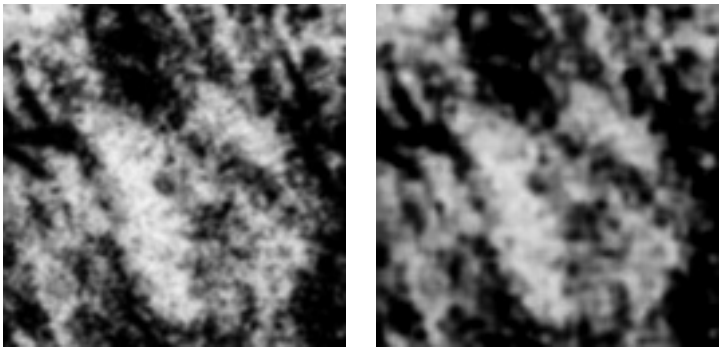


Figure 5.5: Training curve of the autoencoder with best structure. The learning rate was 0.001 and the layers in the encoder part had output dimensions $[64 \times 64 \times 64, 32 \times 32 \times 32, 32 \times 32 \times 32, 16 \times 16 \times 16]$.

The model was tested with some input patches as seen in Figure 5.6.



(a) Input image of autoencoder.

(b) Output image of autoencoder.

Figure 5.6: Input image to autoencoder and corresponding output image.

The output in Figure 5.6b resembles the the input in Figure 5.6a. The noticeable difference is the checkerboard artefacts that is caused by the deconvolutional layers in the decoder part of the autoencoder. This makes the output image look blurred. The autoencoder encoder part was implemented with the processing layers.

5.4 Evaluation of Processing Layers

The structure of the processing layers combined with the encoder part of the autoencoder is firstly based on AlexNet with 5 convolutional layers and 3 fully-connected layers. At the first evaluation this structure will be used to generate the best parameters for learning rate, dropout, activation on output layer and amount of filter kernels in the fifth conv layer. Here the first four conv layers origins from the autoencoder and the fifth conv layer together with the three fc layers are added and only trained on the labelled data. The best networks from the first evaluation and further evaluation is later tested with leave-one-out cross validation on the different rats from the labelled dataset. The first evaluation used the parameters in Table 5.3.

Parameter:	**Description:**
Train set	22068 image patches and encoded signals
Validation set	1154 image patches and encoded signals
Epochs	100
Learning rates	$0.1, 0.01, 0.001, 0.0001$
Layer sizes	$\left[256, 4096, 4096, 360\right], \left[192, 6144, 4096, 360\right], \left[128, 8192, 4096, 360\right]$
Activations on hidden layers	$\left[ReLU, ReLU, ReLU\right]$
Activation on output layer	$None, Sigmoid, Tanh$
Dropout	$0.15, 0.20, 0.25$

Table 5.3: Structures evaluated together with encoder part of autoencoder. Blue numbers are number of convolutional kernels in the convolutional layer.

From this evaluation the best performing parameters will be kept for further evaluation. The structures are evaluated on the validation cost of the last five iterations. This lets more data to be used in training which means that the chances of the networks converging is increased. The validation image patches originates from random images from all rats and image patches from one image is only used in either the training set or validation set never both. The models were trained with the Adam optimizer trying to minimize the MSE cost for each mini-batch[33].

5.4.1 Result of First Evaluation

To measure the performance of the neural network models the cost from the training and validation is stored for every 100 iteration. The training and validation curves of the five best model structures are visualized in Figure 5.7

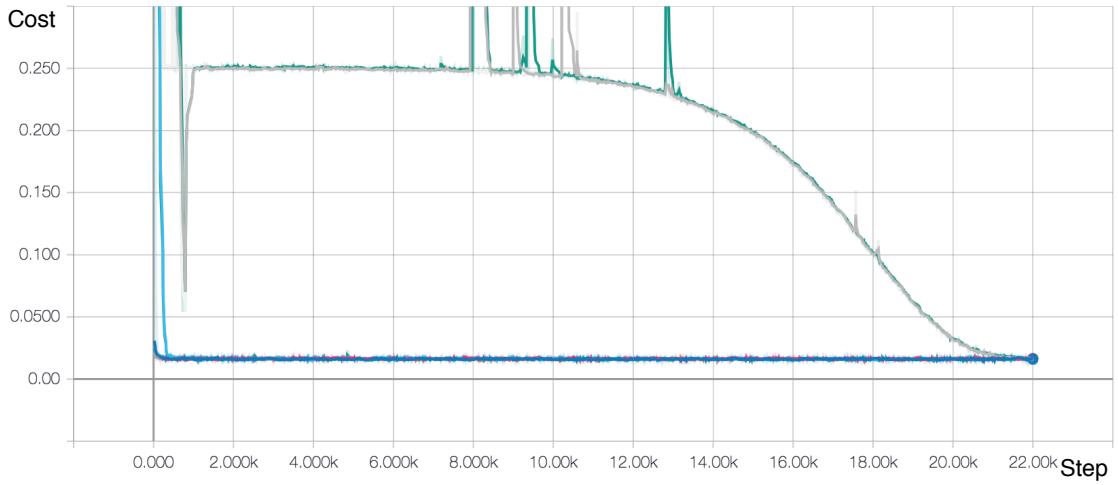


Figure 5.7: The five best models with training and validation curves. The last converging curves, green and grey, are the validation and training curves of structure number 22. The rest of the top structures are located by the blue curves.

Four out of the five best structures converges within the 200 first steps. The last structure with the biggest learning rate converges after 20000 step. The five best structures are shown in Table 5.4.

M:	LR:	DO:	Layer Sizes:	Activations:
112	0.0001	0.15	$[192, 6144, 4096, 360]$	$[ReLU, ReLU, ReLU, None]$
128	0.0001	0.2	$[128, 8192, 4096, 360]$	$[ReLU, ReLU, ReLU, Tanh]$
58	0.01	0.15	$[192, 6144, 4096, 360]$	$[ReLU, ReLU, ReLU, None]$
76	0.001	0.15	$[256, 4096, 4096, 360]$	$[ReLU, ReLU, ReLU, None]$
22	0.1	0.15	$[256, 4096, 4096, 360]$	$[ReLU, ReLU, ReLU, None]$

Table 5.4: Top five structures from first test in descending order with the top being the best.

The cost of the training curves from the best five structures are shown in Figure 5.8.

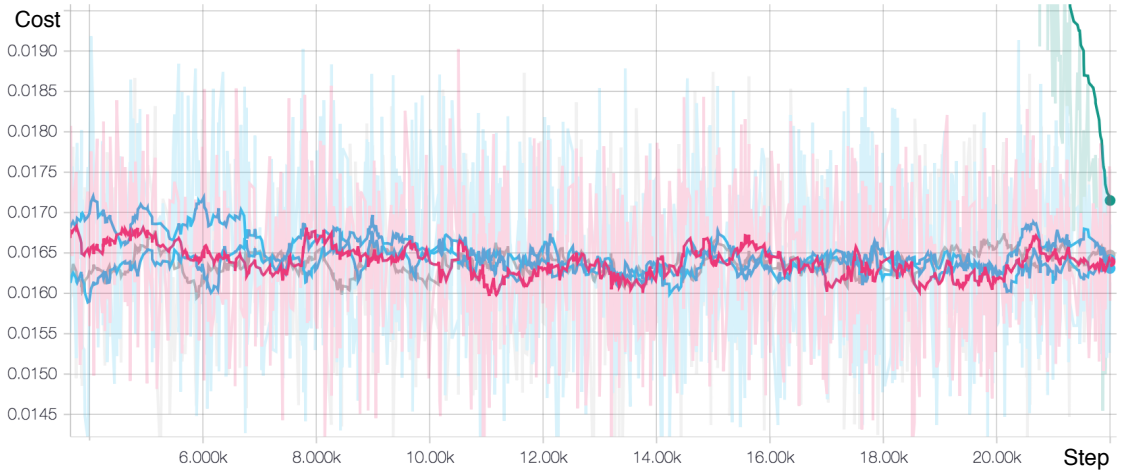


Figure 5.8: Training curves of five best models. Green curve is model 22, grey is 76, top blue is 58, red is 128 and bottom blue is 112.

The curves are smoothed in order to be able to distinguish them. The original values are still visible in the background. From the original values the variance of the curves is big with values ranging from 0.0185 to 0.014. This indicates that there are insufficient amount of training data to train with. The bias is also unsatisfying with a mean cost of 1.65×10^{-2}, which in Figure 5.4 and 5.3 corresponds to a random distribution of points. This suggests that more model structures must be tested. In Figure 5.9, the corresponding validation curves are plotted.

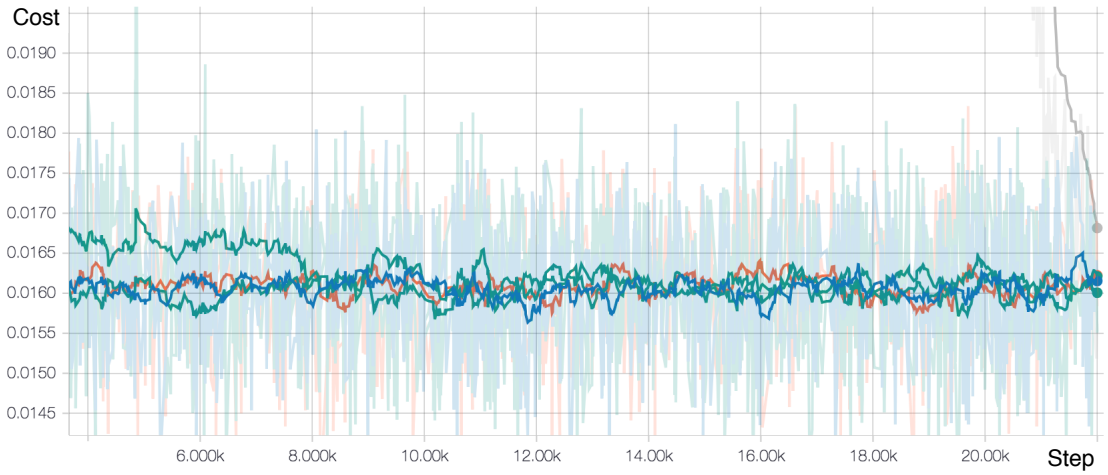


Figure 5.9: Validation curves of five best models. Grey curve corresponds to model 22, top green is 76, orange is 58, blue is 128 and bottom green is 112.

From the validation curves the overall mean cost is lowered from the training, but is still located between 0.0160 and 0.0165. The variance is close to the variance of the training.

In total, the first test evaluated 107 structures with the training and validation set as in Table 5.3. As it can be seen from Table 5.4, four out of the five best performers had no activation on the output layer. The last structure out of those five had $Tanh$ as activation. Also four out of

the five best structures had a dropout in training of 0.15 with the last having 0.20. It can also be shown that both the two best structures had a learning rate of 1×10^{-4}. The biggest learning rate used was the last to converge as it can be seen in Figure 5.7, where structure number 22 is the green and grey curves. Out of the three best structures two had 192 filter kernels in the fifth conv layer.

5.4.2 Further Evaluation

In further evaluation, more layers were added and more combinations with kernel- and layer sizes were evaluated. The best performing parameters from the first evaluation was put as the base for the next evaluation. Further evaluation was done with the new parameters shown in Table 5.5.

Parameter:	**Description:**
Train set	22068 image patches and encoded signals
Validation set	1154 image patches and encoded signals
Epochs	100
Learning rate	0.0001
Number of conv layers	$1, 2$
Number of kernels in conv 1	192
Number of kernels in conv 2	$192, 128$
Tested kernel sizes	$3 \times 3, 5 \times 5$
Number of fc layers	$3, 4$
Neurons in fc 1	$6144, 4096$
Neurons in fc 2	$4096, 2048$
Neurons in fc 3	$2048, 1024$
Neurons in output layer	360
Activation on hidden to output layer	$Tanh, ReLU$
Activation on output layer	$None$
Activation on rest	$ReLU$
Dropout	0.15

Table 5.5: Parameters evaluated in the extended evaluation of processing layers.

The layers are added to the encoder part of the autoencoder again, and evaluated on the mean of the last five costs in validation. In the evaluation 145 different structures were evaluated. Combined with the first evaluation 252 structures were evaluated.

5.4.3 Best Results

The top five results from both evaluations of structures can be seen in Table 5.7.

Model:	Validation cost:
138	1.538×10^{-2}
112	1.539×10^{-2}
128	1.540×10^{-2}
260	1.545×10^{-2}
244	1.546×10^{-2}

Table 5.6: Cost from test set.

The table shows that it is small differences between the best structures. The best performing neural networks and their parameters are shown in Table 5.7.

M:	LR:	DO:	Layer Sizes:	Activations:
138	0.0001	0.15	$[192, 6144, 4096, 2048, 360]$	$[ReLU, ReLU, ReLU, ReLU, None]$
112	0.0001	0.15	$[192, 6144, 4096, 360]$	$[ReLU, ReLU, ReLU, None]$
128	0.0001	0.2	$[128, 8192, 4096, 360]$	$[ReLU, ReLU, ReLU, Tanh]$
260	0.0001	0.15	$[192, 128, 6144, 4096, 2048, 360]$	$[ReLU, ReLU, ReLU, Tanh, None]$
244	0.0001	0.15	$[192, 192, 6144, 4096, 2048, 360]$	$[ReLU, ReLU, ReLU, Tanh, None]$

Table 5.7: Top five structures from first test in descending order. First from top is best. Blue numbers are the amount of filter kernels in the conv layers.

It can be seen from Table 5.7 that the two best structures has very similar structures. The only difference is the number of FC layers.

5.5 Evaluation of AlexNet

In the paper that first suggested the CNNCS model, they demonstrated the state of the art performance with the pre-trained AlexNet model[8]. The model consists of 5 conv layers and 3 FC layers. The last layer is tuned into the size of the encoded signal and has no activation. In Table 5.8, the parameters used in testing of AlexNet is shown.

Parameter:	Description:
Train set	22068 image patches and encoded signals
Validation set	1154 image patches and encoded signals
Epochs	100
Learning rates	$0.1, 0.01, 0.001, 0.0001$
Trainable weights	$True, False$

Table 5.8: Parameters used in the experiment.

In total 8 different AlexNet structures were evaluated. The training set and validation set were

the same as used in the evaluation of A + PL structures. It was also here used Adam optimzer and MSE cost function[33].

5.5.1 Best Results

The trained structures were tested with the same test set as the A + PL structures. The results of this is shown in Table 5.9.

Model:	Validation cost:	Learning rate:	Trainable weights:
8	1.622×10^{-2}	0.0001	$False$
2	1.643×10^{-2}	0.01	$True$
4	1.647×10^{-2}	0.0001	$True$
5	1.652×10^{-2}	0.1	$False$
6	1.667×10^{-2}	0.01	$False$

Table 5.9: Top five structures of AlexNet.

The table shows that with non trainable pre-trained weights and with a learning rate of 0.0001 the AlexNet model performs the best.

5.6 Verification of the Best Result

The best result from evaluations of A + PL and AlexNet were further evaluated. In the previous experiments they were run once over a training set combining images from all the rats. In order to verify the results, the best structures will be evaluated with leave-one-out cross validation, which is illustrated in Figure 5.10.

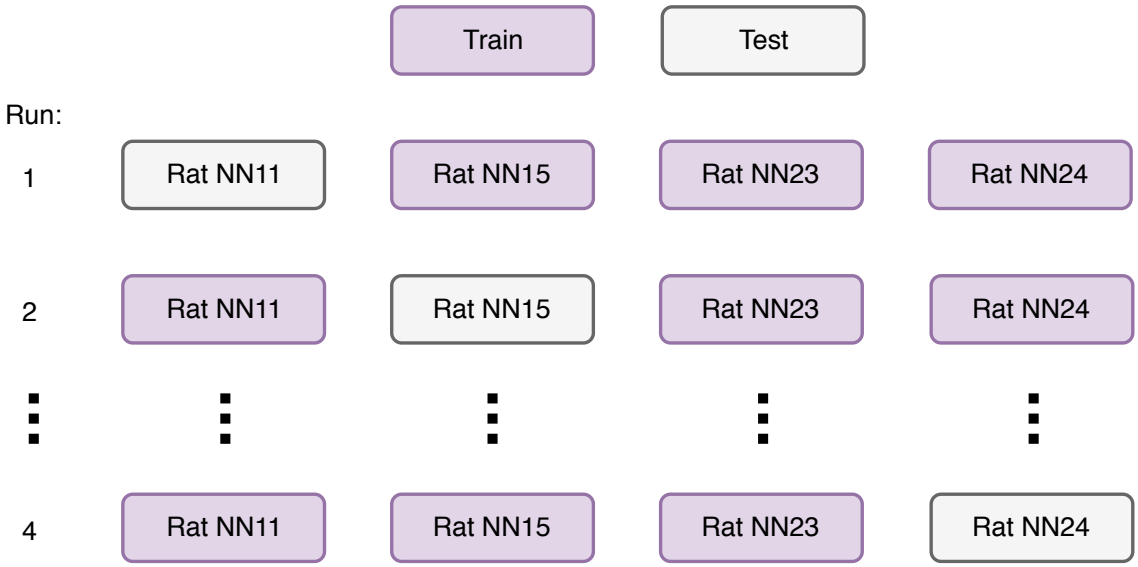


Figure 5.10: Leave-one-out cross validation as conducted in the experiment. The purple boxes corresponds to training data and the grey are used as test set.

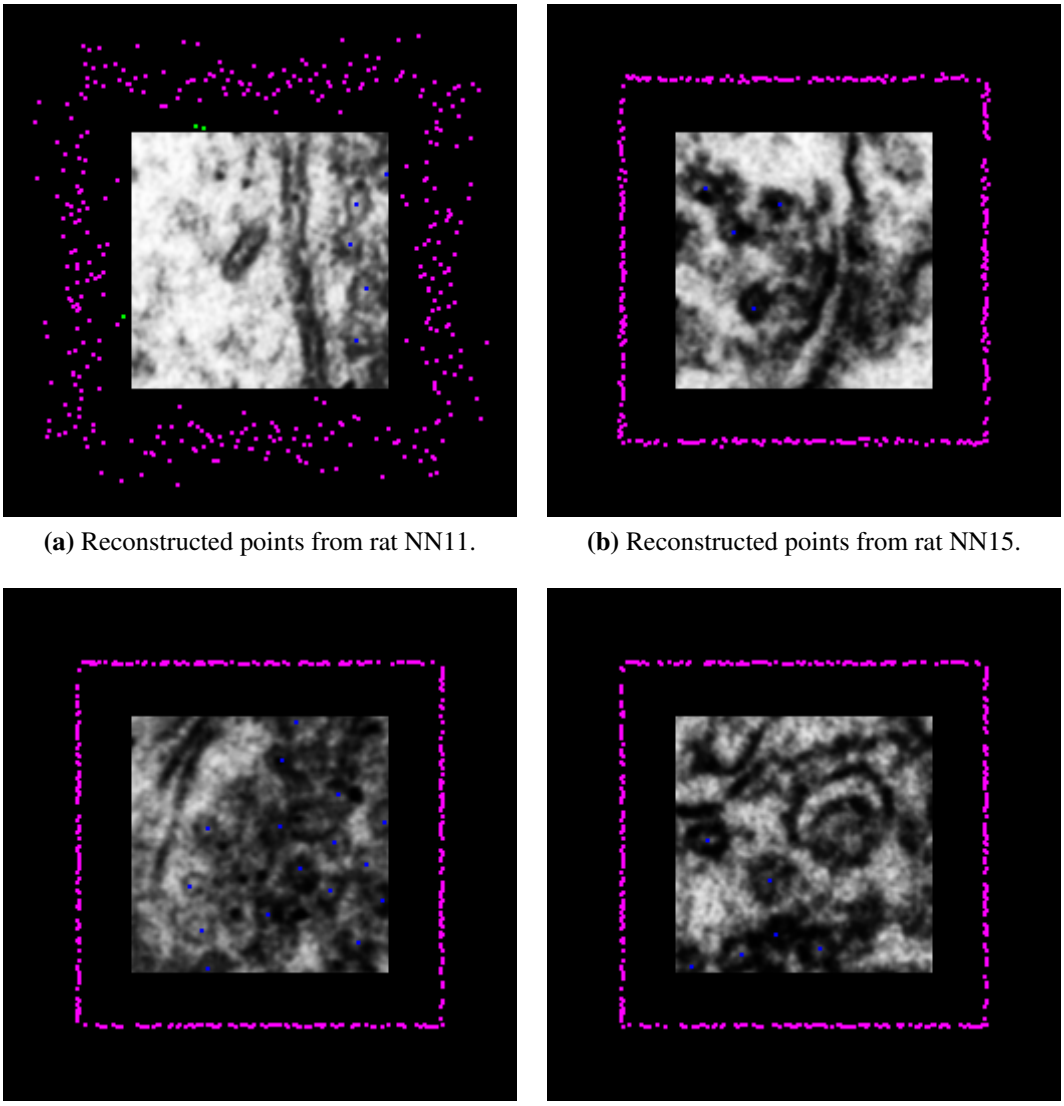
This is done by leaving one rat out as test set and train on the combined images of the other rats. This is repeated until all rats have been in the test set. The performance is then evaluated by the combined results.

Type/Model:	**NN11:**	**NN15:**	**NN23:**	**NN24**	**Total:**
A+PL/138	1.652×10^{-2}	1.387×10^{-2}	1.845×10^{-2}	1.429×10^{-2}	1.611×10^{-2}
A+PL/112	1.652×10^{-2}	1.387×10^{-2}	1.845×10^{-2}	1.429×10^{-2}	$\mathbf{1.610} \times 10^{-2}$
AlexNet/8	1.873×10^{-2}	1.743×10^{-2}	2.116×10^{-2}	1.781×10^{-2}	1.860×10^{-2}

Table 5.10: Results of leave-one-out cross validation.

The results shows that the A + PL structure number 112 has the best overall performance. According to the results the A + PL is also expected to yield better results in general than a pre-trained AlexNet network.

The top performing structure is implemented in the system and produces predicted positions as displayed in Figure 5.11.



(a) Reconstructed points from rat NN11.

(b) Reconstructed points from rat NN15.

(c) Reconstructed points from rat NN23.

(d) Reconstructed points from rat NN24.

Figure 5.11: Reconstructed image points, where blue points are the original vesicle positions and where purple points corresponds to the predicted positions.

The predicted synaptic vesicle positions from Figure 5.11 is not inside the image patch. Instead the visible non-zero elements are close to the location of the observation axes. In Figure 5.11a, the reconstructed images are more noisy than the rest. This is not because of overfitting, but because the biggest rat dataset is used for testing. This means that there are few samples left for training, and the model never learns to approximate the zeros of the signed distances in the encoded signal. Oddly enough, this does not put the cost of the test set into bigger numbers. This may be because of the noisy points acting more similar to the random projected encoded signed distances.

Chapter 6

Discussion

In this chapter the results of the experiments are summarized and further discussed. The overall performance relies on every part of the system functioning in harmony. Addressing which parts that can be held responsible for good or bad performance is a step to improve a future similar system.

6.1 Dataset

A problem concerning most machine learning applications is the amount of available data. In the experiments conducted with neural networks, the training and validation curves are experiencing high variances. This is a indication that the amount of training data is insufficient. Also the data that is annotated only originates from 7 cylindrical cuts from 4 different rat brains. In Figure 3.2 two consecutive images from the same cut is displayed. The consecutive images have small differences because of the brain structure thickness. This can make it difficult for a neural network to map the coherence between changes in the encoded signals to the changes in the images. For processing the images as 2D data the diversity should be greater to make it easier for the neural network to learn the coherence.

If the images from each cut were to be processed as 3D data only the diversity between the cuts would have mattered. But since the synaptic vesicles has a smaller diameter than the thickness between each image they won't appear in two consecutive images. Therefore, 3D processing of the data is not preferable over 2D processing.

6.1.1 Pre-Processing

The goal with the pre-processing was to optimize the data for the neural network. The data was bias corrected to remove the additive bias field. With histogram equalization the synaptic vesicles becomes more distinct in the images. The Gaussian smoothing is added because of the

existing noise in the images.

6.1.2 Data Augmentation

After the image processing, the images were split into 128×128 image patches with and without overlapping. The original images was in different shapes, in order to process them in a CNN with FC layers the images either had to be resized or split into equal dimensions during augmentation. The unlabelled images was split into non-overlapping tiles and further augmented with one random rotation each. The non-overlapping ensured that an increase in similarity between the already similar image patches was avoided. The annotated images were augmented with overlapping due to the insufficient number of images. Due to the fact that only some parts of the images had annotated synaptic vesicles, only the image patches containing annotations were kept. It was proposed to use overlapping of 99.2% or 127 pixels at each translation which would correspond to almost 4.5 million image patches. But as all of these images would originate from the same 68 images the small diversity would not have defended the amount of data. Therefore, overlapping with 87.5% or 112 pixels at each translation was used resulting in 23223 image patches instead.

6.2 Compressed Sensing

Both the encoding and decoding of CS was shown to not generate error in the reconstructed image points with a compression rate up to 2.0. In the decoding process the basis pursuit was the applied minimization problem for the recovery algorithm. However, other methods such as *Lasso* could have been applied but where not looked into in this thesis[34].

6.3 Neural Networks

From the results, illustrated by the figures in Figure 5.11, it is clear that the best neural network model is not able to learn the signed distances in the encoded signal. A great variety of structures were evaluated, but none were able to learn the actual synaptic vesicle positions. This is reflected in the training and validation curves in figures 5.8 and 5.9, where the models struggles to converge.

6.3.1 Autoencoder

Traditional autoencoders are looking for every feature that can be helpful to reconstruct the input and later to classify the total image. In the CNNCS model the goal is not to classify the whole image but to generate predictions on positions of a few elements in it. An autoencoder should therefore be used with caution. When the encoder parts compress the image data with

75% there is a chance that valuable features contained about the synaptic vesicles gets removed in the optimization process of the autoencoder.

6.3.2 Processing Layers

The performance of the processing layers depends on the ability to learn the random projection caused by CS combined with the ability of extracting information of the synaptic vesicles from the autoencoders encoder part. The best model structure has a test cost corresponding to 1.610×10^{-2}, which through experiments conducted on the framework of the system is not satisfactory. In total; 252 different structures were tested. Training without encoder part from an autoencoder quickly turns into overfitting. This indicates that for small datasets it is beneficial to use an autoencoder to prevent the aforementioned problem.

6.3.3 AlexNet

The use of a pre-trained AlexNet with tuned structure was first suggested by Xue and Ray on cell detection using CNNCS[8]. Due to the similarity between their cell images and the synaptic vesicle images it was reasonable that the same structure would be able to predict the synaptic vesicle positions as well. But in reality, with the dataset available it actually performs worse than the A + PL structure. That none of the models are able to accurately predict the encoded signals suggests that the problem is not with the neural network, but with the training data.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The objective of this thesis was to develop an automatic synaptic vesicle detection algorithm that could be used to investigate the impact the stress has on the neural signalling in brains. The developed method is based on a CNN to predict encoded positions of the present synaptic vesicles in the input images. Through the conducted experiments the framework of the system has proven to be robust. However, the CNN has not been able to produce significantly accurate outputs that matches the compressed signed distances. With extended evaluation of different parameters, the best structure matches the encoded signals with a mean squared error cost of 1.610×10^{-2} after leave-one-out cross validation. To be able to predict precise positions of synaptic vesicles, the neural network needs to produce signals with cost from 1.0×10^{-4} and below. The training curve of the CNN model has a very high variance due to insufficient training data and data augmentation has not been of great help. The reason can be low diversity in the available dataset as the images are from only 7 cylindrical cuts in 4 rat brains and each slice has 45 nm thickness preventing sufficient variety in the data.

7.2 Future Work

In future work, it could be investigated if extended transferred learning by pre-training the CNN to match encoded signals from bigger and more diverse datasets could improve the results. An other solution can be trying to impose the sensing matrix in the network structure by extending one of the processing layers into the elements of the sensing matrix.

Bibliography

[1] Wikimedia Commons file:overfitting_svg.svg. https://commons.wikimedia.org/wiki/File:Overfitting_svg.svg. Accessed: 01.05.2018.

[2] D. R. Khalsa. Stress, meditation, and alzheimers disease prevention: Where the evidence stands. *Journal of Alzheimer's Disease*, 48(1):1 – 12, 2015.

[3] M. Khanmohammadi, S. Darkner, Nava N., J.R. Nyengaard, G. Wegener, M. Popoli, and J. Sporring. 3d analysis of synaptic vesicle density and distribution after acute footshock stress by using serial section transmission electron microscopy. *Journal of Microscopy*, 265:101 – 110, 2016.

[4] Shibani Santurkar, David M. Budden, Alexander Matveev, Heather Berlin, Hayk Saribekyan, Yaron Meirovitch, and Nir Shavit. Toward streaming synapse detection with compositional convnets. *CoRR*, abs/1702.07386, 2017.

[5] D. Laptev, A. Vezhnevets, S. Dwivedi, and Buhmann. Anisotropic sstem image segmentation using dense correspondence across sections. *MICCAI 2012, Part I*, pages 323 – 330, 2012.

[6] A. Dahl and R. Larsen. Learning dictionaries of discriminative image patches. *Proc. BMVC*, pages 77.1 – 77.11, 2011.

[7] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *PAMI*, 2016.

[8] Y. Xue and N. Ray. Cell detection with deep convolutional neural network and compressed sensing. 2017.

[9] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski. *Neuronal Dynamics*. Cambridge University Press, 2014.

[10] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Recognition*. Wiley-Interscience, 2000.

[11] J. Nagi, F. Ducatelle, G. A. Di Caro, D. Ciresan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella. Max-pooling convolutional neural networks for vision-based hand gesture recognition. *IEEE International Conference on Signal and Image Processing Applications*, 2011.

[12] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[13] H. Zhao, O. Gallo, I Frosio, and J. Kautz. Loss functions for image restoration with neural networks. *IEEE Transactions on Computational Imaging*, 3(1):47 – 57, 2016.

[14] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249 – 256, 2010.

[15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, (15):1929 – 1958, 2014.

[16] R. H. Wanga, Q. J. Guoa, C. G. Zhua, and C. J. Lib. Multivariate spline approximation of the signed distance function. *Journal of Computational and Applied Mathematics*, 265(12):276–289, 2014.

[17] T. Chan and Wei Zhu. Level set based shape prior segmentation. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2:1164 – 1170, 2005.

[18] A. Dragani, I. Orovi, and S. Stankovi. On some common compressive sensing recovery algorithms and applications. 2017.

[19] G. Kutyniok. Compressed sensing: Theory and applications. 2012.

[20] A. Y. Carmi, L. Mihaylova, and S. J. Godsill. *Compressed Sensing and Sparse Representation*. Springer-Verlag Berlin Heidelberg, 2014.

[21] H. Rauhut, K. Schnass, and P. Vandergheynst. Compressed sensing and redundant dictionaries. *IEEE Transactions on Information Theory*, 54(5):2210 – 2219, 2008.

[22] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790 – 799, 1995.

[23] L. Qu, Y. Akbergenova, Y. Hu, and T. Schikorski. Synapsetosynapse variation in mean synaptic vesicle size and its relationship with synaptic morphology and function. *The Journal of Comparative Neurology*, 514(4):343 – 352, 2009.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25, 2012.

[25] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211 – 252, 2015.

[26] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[27] A. Domahidi, E. Chu, and S. Boyd. ECOS: An SOCP solver for embedded systems. pages 3071–3076, 2013.

[28] E. Jones, T. Oliphant, and P. Peterson. SciPy: Open source scientific tools for Python, 2001–. Accessed: 14.06.2018.

[29] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[30] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[32] Wes McKinney. pandas: a foundational python library for data analysis and statistics.

[33] D. P. Kingma and j. Ba. Adam: A method for stochastic optimization. *International Conference for Learning Representations*, (3rd), 2014.

[34] M. Malloy and R. D. Nowak. Near-optimal adaptive compressed sensing. *CoRR*, abs/1306.6239, 2013.

Appendix

Program Files

Attached in attachments.7z are the program files. 
The python scripts can be described by:

- main_AE_cnncs.py
  - Python script for training neural network with encoder part from autoencoder
- main_alexnet.py
  - File for training AlexNet structures
- main_auto_test.py
  - File for training different autoencoder structures
- main_cnncs.py
  - Code for training CNNCS
- main_cnncs_test.py
  - Code for testing CNNCS
- main_decode.py
  - Python script for reconstructing outputs of neural network into image positions
- main_synth.py
  - Code for training on synthetic data
- cnncs.py
  - Framework file for all training of neural networks
- dataAugmentationRotation.py

  - Code for augmenting data with rotation
- matHistEqSmooth.py
  - Code for histogram equalization and smoothing
- synthVesicle.py
  - Code for making synthetic image patches and corresponding encoded signals
- testClusterData.py
  - Code to make test data for cluster test
- testClusterPlot.py
  - Code for testing post-processing
- testNoise.py
  - Script to making data for noise test
- testNoisePlot.py
  - Code for plotting noise test
- testSensCompress.py
  - Script for compression rate test of CS
- testSensCompressVisual.py
  - Plotting of compression rate test
- testSensSize.py
  - Code to make data for compression rate test
- dilatepoints.py
  - Code to dilate annotations and reconstructed image points
- PY_dataset_sd
  - Folder containing scripts to make image patches and encoded signals for all annotated rats