NTNU

Norwegian University of
Science and Technology

Using Neural Networks to Determine the Origin of Medical Ultrasound Images

Øivind Fossan

Master of Science in Computer Science
Submission date: June 2017
Supervisor: Frank Lindseth, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

This thesis investigates the possibility of using neural networks to determine the body location of medical ultrasound images. Neural networks were trained on several datasets of both synthetic and real ultrasound images, containing labels with the location of each image. Next, the networks predicted the location of unseen images, and the accuracy was measured.

The first dataset consisted of images from three typical locations for ultrasound-guided regional anesthesia, where a classification accuracy of 85.3% was achieved. The second and third datasets consisted of synthetic ultrasound images, and neural networks were trained to predict coordinates for the images, using regression instead of classification. The networks predicted the coordinates with small errors, both in one and two dimensions. The fourth dataset consisted of real ultrasound recordings which covered a large area of the lower arm of nine test persons. Neural networks were trained to predict the 2-dimensional coordinates of the images by regression. In the best case, the average error of prediction was $1.66\ cm$. The final dataset consisted of images of the lower spine of three test persons. A neural network was trained to classify the images as sacrum, vertebra or intervertebral disc, and an average classification accuracy of 66.9% was achieved.

The thesis shows that neural networks can be trained to predict locations of ultrasound images, both by classification and by regression, but more training data and further research is needed to increase the accuracy of the predictions. In addition, the thesis shows that pre-training on natural photographs can be applied to the ultrasound domain to help increase the accuracy of the neural networks.

Sammendrag

Denne masteroppgaven undersøker muligheten for å bruke nevrale nettverk til å bestemme hvor på kroppen medisinske ultralydbilder kommer fra. Nevrale nettverk ble trent på ulike datasett som inneholdt syntetiske og virkelige ultralydbilder, og der lokasjonen til hvert bilde var tilgjengelig under trening. Deretter bestemte nettverkene lokasjonen til nye bilder, og nøyaktigheten ble målt.

Det første datasettet besto av bilder fra tre typiske lokasjoner for ultralydveiledet regional anestesi, der nøyaktigheten på klassifiseringen ble målt til 85.3%. Det andre og tredje datasettet besto av syntetiske ultralydbilder, og nevrale nettverk ble trent i å bestemme koordinater for bildene, ved å bruke regresjon i stedet for klassifisering. Nettverkene fastsatte koordinatene med små feilmarginer, både i en og to dimensjoner. Det fjerde datasettet besto av virkelige ultralydopptak som dekket en stor del av underarmen til ni testpersoner. Nevrale nettverk ble trent i å bestemme 2-dimensjonale koordinater ved hjelp av regresjon. En gjennomsnittlig feilmargin på $1.66\,cm$ ble oppnådd i det beste tilfellet. Det siste datasettet besto av ultralydbilder fra nedre del av ryggraden til tre testpersoner. Et nevralt nettverk ble trent i å klassifisere bildene som enten korsben, ryggvirvel eller mellomvirvelskive, og en gjennomsnittlig nøyaktighet på 66.9% ble oppnådd i klassifiseringen.

Arbeidet utført i denne oppgaven viser at nevrale nettverk kan trenes i å bestemme lokasjonen til ultralydbilder, både ved hjelp av klassifisering og ved hjelp av regresjon, men et større treningssett og videre arbeid trengs for å øke nøyaktigheten til nettverkene. I tillegg vises det at forhåndstrening på naturlige fotografier kan utnyttes for å forbedre nøyaktigheten på nevrale nettverk i ultralydsammenheng.

Preface

This project is the author's master thesis at the Department of Computer Science (IDI), Norwegian University of Science and Technology (NTNU).

I would like to thank my supervisor, Associate Professor Frank Lindseth, for the opportunity to work on this project, and for his helpful advice and guidance over the past months. Also, without his help in arranging and performing ultrasound recordings, this thesis would mostly consist of the introduction, background, and bibliography chapters.

In addition, these contributors were of great help; Dr. Kaj Fredrik Johansen who performed the ultrasound recordings for the regional anesthesia data set, Erik Smistad who helped convert the ultrasound data, and Lars Eirik Bø who assisted during the gathering of the lower spine data set.

I would also like to thank my wife, Heidi, for encouraging me to pursue my degree in computer science, and for her constant support over the last five years. I promise I will not start over again, again.

Øivind Fossan

Trondheim, June 1, 2017

Contents

List of Abbreviations

CNN	convolutional Neural Network
CT	computed tomography
EXP	experiment
GPU	graphics processing unit
LR	learning rate
MAE	mean absolute error
MAEX	mean absolute error in x-direction
MAEY	mean absolute error in y-direction
MR	magnetic resonance
MSE	mean squared error
OCT	optical coherence tomography
ReLU	rectified linear unit
RQ	research question
SGD	stochastic gradient descent
TP	test person

List of Figures

List of Tables

1 Introduction

1.1 Background and Motivation

Since its first applications in the 1940s, the use of ultrasound for medical purposes has been constantly increasing. Today, several imaging modalities with higher image quality are available to medical personnel, such as x-ray-images, computed tomography, and magnetic resonance imaging, but ultrasound remains popular due to its low cost, and its ability to produce real-time images. By placing the ultrasound probe at the location of interest, medical personnel can watch live movements inside the body, like the beating of the heart or the movements of a fetus. Additionally, when surgery or other interventions are performed, the process can be monitored in real time using ultrasound.

A lot of effort is being put into research on automatic interpretation of ultrasound images. However, there is little research on guiding the user to positioning the ultrasound probe at the correct location. Finding the correct location may not be a major problem for specialized medical personnel, but it might be problematic for personnel with less medical training and experience. The methods discussed in thesis are aimed towards such user groups.

Imagine an ultrasound system where the user places the ultrasound probe on the skin, and the system immediately recognizes where on the body the probe is located. Next, when the user specifies a particular feature he/she is interested in looking at, the system tells the user how to move the probe from the current position to find that feature. Like any other automatized ultrasound assistance, such a system could help spread the use of ultrasound technology (Noble, 2016), for example to developing countries with limited resources available for education of medical personnel. Here, the system could be used during training, to train medical personnel at locating certain features, or it could be used during real medical procedures, to assist the medical personnel. Such a system could possibly also be used by medical personnel with little or no experience with ultrasound in emergency situations.

An ultrasound component like this could also play a vital role in a future robotic medical system, which might be fully automatized. To perform anesthetic needle injections, for example, such a system would first have to locate the correct nerve, and this could be done using ultrasound with guidance as described above.

This thesis investigates the possibility of using neural networks to guide the ultrasound probe user to a specified location, using only ultrasound data as input.

1.2 Goals and Research Questions

The goal of this thesis is not to create a fully functional ultrasound guidance system, but rather to begin investigating if neural networks, as a method, has the potential to be useful in such a system. To reach this goal, the following research questions are set forth:

> *RQ1: Are neural networks capable of determining which part of the body an ultrasound image originates from?*
>
> *RQ2: How can neural networks be used to guide the ultrasound probe user closer to a specified body location?*
>
> *RQ3: How accurate can a standard neural network determine the current body location, when using only ultrasound images as input?*
>
> *RQ4: Can pre-training on natural photographs be applied to the ultrasound domain to help increase the accuracy of the neural network?*

1.3 Research Method

To investigate the potential of using neural networks to guide the ultrasound probe user to the correct location, several neural networks were built during the work on this thesis. These networks were first trained on real or synthetic ultrasound images accompanied with labels that described the location the images originated from. Next, the networks were tested on unseen images, in order to predict the image locations. The accuracies of these predictions are some of the main results of this thesis, and the discussion of the results shows that neural networks have potential in the task of guiding the ultrasound probe user.

Five different datasets were used in this thesis. The first dataset was collected and partly studied in a previous study by the author (Fossan, 2016). It consisted of ultrasound images taken at three different body locations – locations typical for ultrasound-guided regional anesthesia (Gray, 2006). Here, a neural network was trained to classify new images as belonging to one of these three locations.

The second dataset consisted of 37 simple, synthetic images, produced for this thesis. These images simulated a linear direction ultrasound recording, with two features that changed predictably over the recording. A neural network underwent regression training, to predict where in this recording sequence new images originated from.

The third dataset also consisted of synthetic images, produced for this thesis. It was an extension of the second dataset, to include probe movements in two dimensions, and to include multiple recordings with variations. The dataset was supposed to simulate several linear direction ultrasound recordings, with variations in lateral offset, and taken from different persons. A neural network was trained to predict the location of new images in two dimensions.

The fourth dataset consisted of real ultrasound images taken on the lower part of the arm of nine people, with varying lateral offset. This was a continuation of the tests done on the third dataset, to see if the method worked on real ultrasound images.

The last dataset contained images from ultrasound recordings of the lower spine of three persons. The recordings were started at the sacrum and extended over the five lumbar vertebrae, and the images in the dataset were given one of three labels: sacrum, vertebrae, and intervertebral disc. A neural network was trained to predict these classes.

1.4 Contributions

The contribution of this thesis to the fields of neural networks and ultrasound, is to show that neural networks can be trained to predict the acquisition location of ultrasound images by using only raw image data as input, either through classification or through regression. As no previous studies were found to demonstrate this method, this thesis could encourage more research on this area.

1.5 Thesis Structure

The thesis is divided into six chapters. The first chapter presents background and motivation for the thesis, in addition to goals and research questions. Chapter 2 provides an introduction to neural networks and to ultrasound, and presents relevant previous work in these fields. Chapter 3 describes the methods used in this thesis, including descriptions of the datasets, neural networks, and settings used in each experiment. Chapter 4 presents the results of the

experiments, and chapter 5 discusses the results. Chapters 3, 4, and 5 are sub sectioned into five groups of experiments, corresponding to the five datasets used in the thesis. Finally, a conclusion is given in chapter 6, and future work is suggested in chapter 7.

2 Background

This chapter will first present a theoretical introduction to neural networks, with emphasis on the aspects that are relevant for this thesis. In section 2.2, a similar introduction is given to the field of ultrasound. Section 2.3 presents a structured literature review, which was conducted to find the most relevant studies on the combination of neural networks and ultrasound. The studies resulting from this review are presented in section 2.4. Finally, section 2.5 provides a baseline to compare the results of this thesis against.

2.1 Neural Networks

Neural networks is a field that currently attracts great interest and is being researched extensively. There are many good introductory resources to the field, such as Goodfellow, Bengio, and Courville (2016), and LeCun, Bengio, and Hinton (2015). This section aims to briefly introduce some of the most important concepts of neural networks, relevant for this thesis.

2.1.1 General

Machine learning is a computer science subfield that enables computers to learn their behavior automatically, instead of having their behavior programmed explicitly. The learning can be performed in a *supervised* or *unsupervised* manner. Supervised training allows the program to experience input data examples along with their correct output during training. In unsupervised training, the program instead tries to learn useful properties or structures only by experience the data examples (Becker & Plumbley, 1996).

One of the branches of machine learning is artificial neural networks, which was originally inspired by biological neural networks, although the similarity is rather limited. As with many other machine learning techniques, standard neural networks are fed with data (called examples) along with the correct output for these examples (called labels). The neural network is then trained to find a suitable mapping between the given examples and the desired label. The goal is that, after the training is complete, the neural network will be able to produce correct labels from new, unseen examples as well. The training of neural networks in this thesis is done in a *supervised* manner.

2.1.1.1 Classification and Regression

Neural networks can be used to solve many different categories of problems (Goodfellow et al., 2016, p. 99). Two of these categories, which are applied in this thesis, are classification problems and regression problems. In a classification problem, there is a predefined set of classes that each example can belong to, and the task is to find out which class each example belongs to. In a regression problem, each example is associated with a real number (or several numbers), and the task is to predict this real number for each example.

2.1.1.2 Generalization

An important quality of a neural network is how good it is at generalizing (Nielsen, 2015, chap. 3). Generalization is the ability to apply the knowledge gained from labelled training data to new examples. To do this, the network must learn what the essential features in the data are, and how to recognize these features. When the same features are recognized in new examples, they provide the network with clues to the correct classification, or to the correct regression value. For example, if a network is trained to classify different animals, learning to recognize features like elephant trunks helps the network classify new elephant images correctly.

The opposite of generalization is overfitting, which is a common problem in neural networks. When a network overfits, it is trained *too well* on the examples, to the point where it starts memorizing the examples instead of learning the key features. A network that memorizes the training examples is very good at guessing the correct labels for those training examples, but performs worse when given new examples, since they are not identical to the memorized examples.

2.1.1.3 Training, Validation, and Test Sets

To reduce the effect of overfitting, it is common to test the performance of a neural network on different examples than the ones used during training. Typically, the available examples are divided into three sets (Goodfellow et al., 2016, pp. 110, 120): A training set, a validation set, and a test set.

The training set contains examples that are used by the network during training, along with the correct labels for each example. After the training process, the unseen examples of the

validation set are presented to the network, and the accuracy of classification or regression is measured. This is done since measuring the performance on the already seen examples gives little information of how good the network is at generalizing. Typically, after this measurement, changes to the network's structure or hyperparameters are made (see section 2.1.5), and the training process is repeated. Next, the performance on the validation set is measured again, and compared to the previous validation set performance. This process can be repeated many times until the optimal hyperparameters are found. At this point, a final test of the network can be made on the examples of the test set. This is necessary, since the above iterations of training and testing may have biased the network towards performing well on those particular validation set examples. Testing it on a third test set gives better information about how good the network is at handling unseen data.

Although this partitioning gives the most accurate performance indication, it is also very common to only split the examples into two sets – a training set and a test set – as this requires fewer examples and is less time consuming. The latter strategy is applied in this thesis.

2.1.1.4 Cross-validation

Ideally, a neural network is trained on thousands or even millions of different examples to learn the essential features and to generalize well. However, sometimes, e.g. while doing research, the access to training data may be very limited. An example can be the case of recognizing handwritten characters (LeCun, Cortes, & Burges, 2016); if the researcher only has access to handwritten characters from five people, of which four are used for training and one is used for testing, the classification accuracy achieved may not be a statistically good measure of how the network would perform on unseen data. If the researcher has several different neural network models to choose between, he/she may not feel confident about which one is best suited for the task, based on the small number of examples.

In such cases, the cross-validation technique may be useful (Arlot & Celisse, 2010) to make the results statistically more valid. In its most common form, called leave-one-out, the cross-validation technique consists of setting aside one example as the test set, and training on the rest of the examples. This is repeated several times, but with a different example in the test set each time, until all examples have been in the test set. The average performance over these runs

is the cross-validation performance, and it gives a better indication of how the network will perform on new data than by simply training and testing once.

2.1.1.5 Imbalanced data

Imbalanced data are training data that are unevenly spread over the possible classes of a classification problem (or over the possible values of a regression problem). The effect of this is that the network will spend more time training on some classes (or some range of values) than on others, which can make the network biased towards predicting those classes. Imbalanced data sets are very common; for example, He and Garcia (2009) discuss the severely imbalanced, and widely used "Mammography Data Set", a collection of mammography images from distinct patients with binary classification. In this data set, 10 923 images are classified as "cancerous" (the majority class), and 260 images are classified as "healthy" (the minority class, about 2% of the images). He and Garcia found that classifiers tended to produce a highly imbalanced degree of accuracy on the different classes. The images from the majority class tended to have close to 100% accuracy, while the images from the minority class typically were classified correctly only 0-10% of the time, meaning that about 234 cancerous patients were incorrectly classified as healthy. The paper suggests several ways to combat this effect, but the most common way is to apply a sampling method that makes the data set behave as a balanced set. For example, the sampling method could randomly pick an equal number of images from all classes during each iteration of the training process. Another solution is to choose examples from the different classes with uneven probabilities. A third option is to use data augmentation to add examples to the underrepresented classes (see section 2.1.4.1).

2.1.2 Architecture

A neural network consists of several layers of neurons – also called nodes. Nodes in one layer are connected to nodes in the next layer, while there are no connections between nodes within the same layer. The first layer is called the input layer. No calculations take place in this layer – it is merely the entry point to the network where examples are input, one value per node. The last layer is called the output layer, which is where the result of the network's computations can be found. All the other layers are called hidden layers, since they are normally not accessed externally. When a neural network is fed with data in the input layer, the values are forwarded

to one or more nodes in the next layer. These nodes perform calculations on the values they receive, and forward the results to one or more nodes in the next layer, which repeat the process.

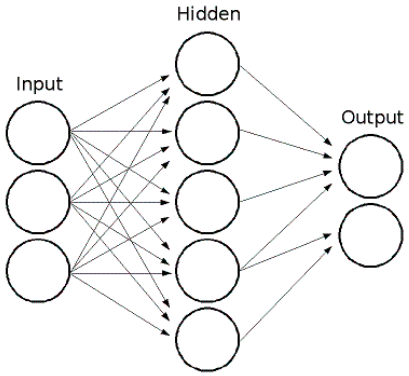

Figure 2.1: A typical neural network architecture

An illustrative neural network is seen in Figure 2.1. This network is called a two-layer network as input layers are not counted as complete layers. Here the input layer consists of three nodes, there is a single hidden layer consisting of five nodes, and the output layer consists of two nodes. All nodes in the first layer are connected to all nodes in the second layer, and this makes layer 2 a fully-connected layer. On the other hand, in this example, the output nodes only receive input from some of the nodes in the hidden layer.

Each hidden node and output node performs calculations. The inputs to these calculations are the values received from nodes in the previous layer, along with one weight per value. The weights are multiplied by the values, which determines how much influence each value has on the calculation. The products of these multiplications are summed and passed through an activation function, which also takes a single bias as input. One such activation function is discussed in section 2.1.2.1. The output of the function is a single number that is passed on to nodes in the next layer.

In the context of image analysis, the input to the NN would be an image and each image pixel value is given to a separate input node. The next layer then performs calculations on these pixel values, and produces a vector of values, which can be seen as a new representation of the image. Conceptually, each layer creates a more abstract representation (Goodfellow et al., 2016, p. 6) of the original image. For example, the first layer may represent the image as simple shapes like dots and corners, the second layer may connect these shapes into more complex shapes like contours, and so on. Finally, the output nodes may be able to classify the image, or to produce

a regression value, describing some property about the image. Typically, when used for classification purposes, each output node corresponds to one of the possible classes, and the output node with the highest value indicates the network's classification of the image.

When training a neural network, the task is to determine the appropriate weights and biases, so that the network produces the desired output. This process must be performed automatically, since most networks are very large, containing millions of weights, and setting the values of these manually is infeasible.

2.1.2.1 Rectified Linear Unit

There are many available choices for the activation function within each node. However, to make the neural network capable of approximating any function, the activation function has to be non-polynomial (Leshno, Lin, Pinkus, & Schocken, 1993). The default recommended activation function today is the ReLU (Glorot, Bordes, & Bengio, 2011; Jarret, Kavukcuoglu, Ranzato, & LeCun, 2009; Nair & Hinton, 2010). The ReLU function is given below:

$$h(a) = \max(0, a), \quad \text{where}$$

$$a = Wx + b$$

In these formulae, a is the product of the previous layer's output vector x times the weight matrix W, plus a bias b vector associated with the nodes. The max function ensures that the ReLU function is non-polynomial.

2.1.2.2 The Softmax Function

When a neural network is used for classification, the softmax function is often used as an activation function of the output layer (Goodfellow et al., 2016, p. 183). This makes the values of the output layer behave like probabilities, by summing to one while still reflecting the relative differences between the values. Consequently, after applying the softmax function, an output value can be interpreted as the network's belief in a certain example belonging to a certain class; e.g. if one output node gets a value close to one, the network is almost 100% sure that this is the correct classification of the example.

The softmax function is

$$\sigma(\boldsymbol{x})_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}, \text{ for } j = 1, \dots, K$$

, where $\boldsymbol{x}$ is the original K-dimensional output vector. The summation in the denominator causes the resulting softmax values to sum to 1.

2.1.2.3 Convolutional Layers

In the above description of neural networks, fully-connectedness was presented as a common way to connect two layers. However, when the input to the network consists of images, the use of convolutional layers is very common. Any network that includes convolutional layers is called a *convolutional neural network*, or CNN (Krizhevsky, Ilya, & Hinton, 2012).

Typically, in a CNN, the nodes of all layers except the last two or three layers are arranged in a two-dimensional grid, since the image input is also two-dimensional. In a convolutional layer, a node is not connected to all nodes of the previous layer. Instead, the value of a node is determined by convoluting a small filter with the corresponding nodes in the previous layer. For example, if a filter of size 5×5 is used, each node in the convolutional layer will be influenced by the 25 closest nodes in the previous layer, called the receptive field. Restricting the influence in this way makes sense, since spatial proximity is an important part of image interpretation.

Also, the same filter is used by all nodes in the convolutional layer, which means, in the example above, that only 25 weights will have to be learnt. This sharing of weights significantly reduces training time, when compared to a fully connected layer which can contain thousands of weights.

By adjusting the weights, a filter can be trained to recognize certain features anywhere in the image, like an edge in the early layers, or a body part in the later layers. For example, a filter with weights tuned to detect vertical edges will cause a large activation value when placed on top of such an edge. The output of applying such a filter to a layer is a grid of values indicating the belief in the presence of vertical edges at each pixel.

Naturally, since there might be more than one interesting feature to look for in each layer, multiple filters are normally used per layer. When k filters are used on a $n \times m$ layer, the next layer will contain k parallel layers of size $n \times m$. When multiple convolutional layers are stacked on top of each other, this causes the number of weights to grow quickly. The next section describes a technique to compensate for this increase.

2.1.2.4 Max-pooling

A convolutional layer is often followed by an activation function and a max-pooling layer (Goodfellow et al., 2016, p. 339). The purpose of the max-pooling layer is to provide a more compact representation, by creating a statistical summary of the outputs from the previous layer. In this process, the layer size is downscaled, so that the increase in the number of weights caused by the parallel filters of the convolutional layer is somewhat compensated for.

To create the information summary, a filter, e.g. of size 2×2, is moved across the nodes of the previous layer with a certain stride (normally the width and height of the rectangle). For each location of this rectangle, the maximum reported value is found, and this value is transferred to a single node in the max-pooling layer. Consequently, the max-pooling layer will be much smaller than the previous layer; in the case of a filter of size 2×2, and with stride 2 in both dimensions, the max-pooling layer will be half as high and half as wide as the previous layer.

The effect of the max-pooling operation is that each node in the max-pooling layer indicates the maximum belief in the presence of a certain feature within a certain area. To some extent, this loses spatial information, since after the max-pooling operation, it is unknown exactly where a node's value comes from. At the same time, this produces an invariance to small local translations, which helps increase generalization. Often, it is more important to recognize that a certain feature *is* present in the image, than to know exactly where this feature is located.

2.1.3 Training

2.1.3.1 Loss Functions

In order to adjust the weights of a network automatically during training, a measure of network quality is needed; the network needs to know whether changing a weight in a certain direction is beneficial or not. This quality measure comes in the form of a loss function (also called cost function), which measures how *badly* the network performs (Goodfellow et al., 2016, p. 177). Therefore, the goal of the training process is to minimize the loss function.

After the training phase, the same or another loss function can be run on the network to get a measure of the final quality of the network when introduced to new examples. In this thesis, these functions are called test functions, although the functions can be identical to the ones used as loss functions. The loss/test functions used in this thesis are described below.

Cross-entropy

When the network is used to solve a classification problem, the *cross-entropy* (Nielsen, 2015, chap. 3) is a common choice of loss function:

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log\left(prediction_{ij}\right) \tag{2-1}$$

In this function, N is the size of the training set, y_i is the correct label of the i^{th} example, and $prediction_i$ is the predicted label of the i^{th} example, produced by the network. A label may contain several entries, and in the case of classification problems, the label is normally one-hot encoded. This means that the number of entries in each label is equal to the number of possible classes. The label entry corresponding to the correct class is given the value 1, while all other entries are given the value 0. In the above function, y_i and $prediction_i$ are therefore both vectors with size equal to the number of possible classes, M. The inner summation adds together the cross entropy of each pair of items from the two vectors by using the log-function, such that a large difference between the labels will produce a large negative number, while a small difference will produce a negative number close to zero. The outer summation is over all the examples in the set. Finally, the sign is changed, and the result is divided by N to get the average

loss over all the examples. In summary, if the network is able to predict labels that are close to the correct labels, the cross-entropy will be small.

Mean Squared Error

The *mean squared error* (MSE) is a measurement of the average of the squares of the differences between values. One effect of squaring the differences is that large differences are weighted more than small differences. Since the MSE is concerned with real numbers instead of categories, it is better suited for regression problems than for classification problems. In the context of neural networks, this means that the examples have real numbered labels, and the task of the network is to predict the values as accurately as possible. The MSE function is shown below:

$$\frac{1}{N}\sum_{i}(y_i - prediction_i)^2 \tag{2-2}$$

Here, y_i is the correct label, and $prediction_i$ is the predicted label of the i^{th} example of the training set of size N. The difference is squared and averaged over all N examples, to give the loss of the current network, which is to be minimized during training.

In the case where each label is a vector of M numbers instead of a scalar, the MSE can be extended as shown below:

$$\frac{1}{MN}\sum_{i}\sum_{j=1}^{M}\left(y_{ij} - prediction_{ij}\right)^2 \tag{2-3}$$

The inner summation is over the squared error of each of the M components of a label, while the outer summation is over all examples of the training set. The total sum is divided by the training set size N and the number of components in each label M, to produce the combined mean squared error.

Mean Absolute Error

Like the MSE, the *mean absolute error* (MAE) is also applicable when comparing real valued numbers, but instead of squaring the differences, the absolute value is used, as shown below.

$$\frac{1}{N}\sum_{i}|y_i - prediction_i| \tag{2-4}$$

Here, y_i is the correct label, and $prediction_i$ is the predicted label of the i^{th} example of the training set of size N. In contrast to the MSE, the output of the MAE is of the same scale and unit as the input, and can be interpreted as the average difference between the predicted values and the correct labels.

Equation 2-5 shows a simple extension of the MAE to situations where the labels are vectors of M numbers instead of scalars.

$$\frac{1}{MN}\sum_{i}\sum_{j=1}^{M}\left|y_{ij} - prediction_{ij}\right| \tag{2-5}$$

2.1.3.2 Optimization

Optimization (or learning) is the process of updating the weights and biases during training to minimize the loss function. To accomplish this, a technique called *stochastic gradient descent* (SGD) is applied (LeCun et al., 2015, p. 437), which computes the effect on the loss function of changing every single weight, in the context of some or all of the training examples. The details of this computation, called back-propagation (Nielsen, 2015, chap. 2), are not covered here, as they are rather lengthy and technical. The result of the SGD is a vector of optimal weight changes, called the gradient. This gradient is added to the current weight values, hopefully bringing the network one step closer to learning an approximation of the problem's underlying function.

The fastest way to compute the gradient is to only consider a single example from the training set. The gradient will then consist of weight changes which will cause that particular example to be predicted more correctly. However, when sweeping through the training set, some

examples may cause weight changes in one direction, while others may cause weight changes in the opposite direction. A more accurate gradient can be found by regarding all examples simultaneously, so that adding the gradient will cause the predictions on the training set as a whole to be better. The downside of this is that it takes a long time to perform a single update on the weights, resulting in a long total training time. The compromise between these two extremes is to consider a mini-batch of the training examples in each update, as an approximation of the whole training set (Goodfellow et al., 2016, p. 150). The examples in the mini-batch are stochastically chosen for each iteration.

Several variations of the SGD have been proposed and many are in use today. One of these, which is used in this thesis, is the Adam optimizer (Kingma & Ba, 2015).

2.1.4 Regularization

To combat the effect of overfitting, several regularization techniques have been developed (Goodfellow et al., 2016, p. 228). The techniques relevant for this thesis are described below.

2.1.4.1 Dataset Augmentation

One of the best ways to make a neural network generalize well, is to provide it with a huge training set (Goodfellow et al., 2016, p. 459), so that the network will not train repeatedly on the same examples, but rather be forced to learn the essential features to correctly classify an example. However, when the access to training data is limited, as is often the case, a technique called dataset augmentation can be useful. Dataset augmentation expands the training set artificially, by making modified copies of the examples already present. These small modifications may include translation, rotation, scaling, skewing, and flipping, and other suitable modifications. This can help the network generalize, since it can learn to detect a certain object at many locations in the image, with varying degree of rotation, etc.

2.1.4.2 Dropout

One way to combat overfitting is to first train many networks with varying architecture and weight initialization, and then, during testing, take the average of the predictions from all networks as the final prediction. However, the memory usage and computation power needed

in this approach quickly becomes too large. Instead, a very popular regularization technique called *dropout* (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014) is often applied. Dropout mimics the approach described above in a single neural network. The main idea is to randomly deactivate (or drop) about half of the nodes in each iteration of the learning process. Thus, in each iteration, the network can be seen as a thinned version of the original network, which will be different each time. Because of this, the nodes in the network are forced to not rely on the presence of certain other nodes. Instead, they must learn to predict the correct label by looking at a range of evidence, since any single evidence may be absent. The result of this is a more robust, and less overfitted network.

2.1.4.3 L^2-regularization

Another common regularization technique is called L^2-regularization (Goodfellow et al., 2016, p. 231). In this approach, the loss function is extended by a new term that penalizes large weights in general. Because of this, weights that contribute a lot to the quality of the network can be increased somewhat, while other weights will be driven towards zero. Forcing the weights towards zero is beneficial, since small weights make the network less sensitive to small changes in the input, thus increasing generalization.

2.1.4.4 Transfer Learning

The standard way to train a neural network is to initialize the weights according to a certain probability distribution, and then start tuning the weights to lower the loss function. The learning process often takes a long time, but experiments show that the training time can be significantly reduced by utilizing the concept of *transfer learning* (Yosinski, Clune, Bengio, & Lipson, 2014). The main idea of transfer learning (also called pre-training) is that the features learned in the early layers of neural networks are often quite similar from network to network, even though the problems to be solved may differ considerably. Therefore, instead of randomly initializing all weights, the weights in the early layers may be copied from another network that is already trained. Although these weights may be changed significantly during the subsequent training process, it is often a much better starting point than training from scratch.

The number of layers to copy weights from is not given; this can be experimented with in each application. Also, the pre-trained layers may either be frozen from further training, or included

in the training process. Freezing the layers speeds up training time, but at the same time limits the networks ability to improve prediction quality.

Transfer learning also provides regularization, especially in cases where the training set is small, and the risk of overfitting is high. In such cases, a network can first be trained on a large, publicly available set, and then trained further on the smaller original dataset, which reduces overfitting considerably.

Another approach to transfer learning is to pass the training examples through some pre-trained layers, and store the output as a new representation of the examples. Since the pre-trained layers already have learned to detect certain basic features, the presence of these features will be included in the new representation. These examples can then be used as input to another neural network.

2.1.5 Hyperparameters

As described in section 2.1.1.3, hyperparameters are parameters that control the overall training process. A typical workflow is to first train the network and record the performance. Next, a hyperparameter is changed, and the training is restarted. Comparing the performances of the two runs reveals if the change was beneficial or not. This process can be repeated until an optimal choice of hyperparameters is found. Some common hyperparameters, relevant in this thesis, are described next.

2.1.5.1 Number of Epochs

When all training examples have been considered by the SGD, one *epoch* is said to be completed (Goodfellow et al., 2016, p. 246). Normally, the network is trained for many epochs, and the performance is gradually improved over the epochs. However, training for too many epochs can cause overfitting. Since the network sees the same examples repeatedly, it may begin memorizing the examples. The number of epochs is therefore a compromise between training long enough, and not training for too long.

The overfitting effect normally becomes visible when the network is tested on another set of examples. Therefore, one way to determine the number of epochs is to test the network on the

test set after each epoch. When the performance on the test set begins to decrease, the training process should stop.

2.1.5.2 Batch Size

As described in section 2.1.3.2, the batch size (the number of examples considered simultaneously) can greatly affect the training time and the performance of the network. Choosing a too large batch size will slow down the learning, since the weights are only updated once per iteration of the SGD. On the other hand, choosing a too small batch size may cause the resulting gradient to be far from optimal. The best value for the batch size is data dependent and should be experimented with in every application.

2.1.5.3 Learning Rate

After the SGD method has computed the gradient, the gradient is added to the current values of the network's weights. However, the weight changes in the gradient are not perfect, since they are based on a mini-batch of the training set. Even if they were based on the entire training set, they would still be imperfect, since the training set approximates all possible examples. Because of this, when adding the gradient, some weights will be moved closer to their optimal value, while other weights may be moved too much, overshooting the optimal value. Because of this, the gradient is multiplied by a small number – the learning rate λ – before being added to the weights (Goodfellow et al., 2016, p. 151). A small learning rate will therefore cause slow, but careful learning, while a large learning rate will speed up training, at the risk of overshooting the optimal weight values.

Choosing the best learning rate is often done by experimentation, as it is dependent on the problem and data at hand.

2.1.5.4 Learning Rate Decay

Related to the concept of learning rate is *learning rate decay* (Goodfellow et al., 2016, p. 294), which controls the rate of change of λ (learning rate). It is often desirable to have a large λ in the early epochs, and then gradually decrease λ as the training progresses. The reason for this is that even if we arrive at the optimal weights, the random sampling of the mini-batches will

always cause the gradient to be larger than zero, so that adding the gradient will move the weights away from the optimal values. With learning rate decay, if λ is large first, and then gradually reduced, we ensure that the update of the weights will be smaller and smaller as we approach the optimal values.

2.2 Ultrasound

This section will briefly present key concepts related to ultrasound (Chan & Perlas, 2010), since most of the images used as input to the neural networks in this thesis are ultrasound images.

2.2.1 Basic Concepts

Ultrasound is a medical imaging technology used to produce internal body images. The images are produced by sending sound waves into the body and listening for the response.

The ultrasound equipment typically consists of an ultrasound probe, connected to an ultrasound machine which contains various controls and a screen. The probe acts as both a transmitter of sound waves and a receiver of echo waves. Both the transmission and the reception of sound waves are performed by transducers placed inside the probe. These transducers utilize the piezoelectric effect (Manbachi & Cobbold, 2011), which converts electric signals into mechanical energy, and vice versa. The ultrasound machine sends electric signals to the probe, where they are converted into sound waves that are sent into the body. When the echoes of these sound waves are returned, they are converted to electric signals, which are passed on to the ultrasound machine. The ultrasound machine then interprets these signals, and generates an image on the screen.

As the sound waves travel through the body and encounter different kinds of tissue, some of the energy is reflected, while some of the energy continues forward. This will produce a series of echoes that eventually reach the ultrasound probe. By measuring the time from the original sound wave was transmitted until the different echoes return, the distance (or depth) to the origin of the echoes can be calculated.

If v is the speed of sound and t is the time from transmission until an echo returns, the distance d to the origin of the echo can be computed as

$$d = \frac{vt}{2}$$

In this equation, the right side is divided by two, since the sound must travel both forth and back.

Echogenicity is a tissue's ability to produce an echo. The different tissues in the body have different echogenicity, which means they will produce echoes of varying strength. Some tissues, like bone, reflect almost all the energy, while very soft tissues, like water, let all the energy pass through it. By analyzing the strength of the echoes received, the internal structure of the body can be visualized, for example by letting strong echoes show as bright dots on the screen, while weak echoes show as dark dots. Since the ultrasound probe usually transmits sound in different directions simultaneously, or produce many parallel sound waves, a two-dimensional image can be synthesized from the information in the echoes.

Since some of the mechanical energy is reflected, the sound wave is attenuated as it travels deeper into the body. Therefore, echoes produced near the surface are naturally stronger than echoes produced deeper inside the body. This effect must be corrected for when creating the final image.

2.2.2 Assumptions

The ultrasound equipment makes several assumptions to produce the final image. One of these is that the speed of sound is constant inside the body – $1540\ m/s$. In reality, the speed of sound varies a lot depending on the medium it travels through. The speed is slower through fat ($1450\ m/s$) and air ($330\ m/s$), and faster through blood ($1570\ m/s$) and bone ($4080\ m/s$) (Stern, 2016). Nevertheless, the assumption of a constant speed is good enough to produce images, partly because air and bone have a very high echogenicity, so the sound waves never travel far into these tissues at all. Another effect of the high echogenicity is that bone and air produce shadows in the image, so that features behind them become invisible.

The second assumption is that sound travels in a straight line. Although this is not entirely true, the assumption works for distances under $20\ cm$ (Holm, 2008). This also means that the

ultrasound probe can send sound in several directions simultaneously, and expect echoes to return in the opposite directions, enabling the equipment to stitch together many strips of data into a two-dimensional image.

The third assumption is that the sound wave attenuation is constant inside the body, which is also not true. The degree of attenuation varies with tissue type, but, again, the assumption is good enough to produce images. Also, the attenuation highly depends on the sound wave frequency. High frequencies cause a lot of attenuation, but at the same time, high frequencies are desirable because they increase the spatial resolution of the images. This is partly the reason why the depth of ultrasound images is very restricted.

2.2.3 Advantages and Disadvantages

One of the main benefits of the ultrasound technology over other imaging modalities, like magnetic resonance (MR) or computed tomography (CT), is that the ultrasound equipment can produce images in real time. This makes the technology ideal for monitoring surgery, and watching live movements of fetuses, among other things. Another advantage over CT is that ultrasound is risk-free, as the body is exposed to sound waves instead of radiation. In addition, ultrasound is cheap and portable, compared to both CT and MR.

The main disadvantage of ultrasound is the poor quality of the images, because of the many assumptions that must be made. The image resolution can be low, the images are often noisy, and bone and gas prevent certain areas from being examined by ultrasound.

Further information about the development of the ultrasound technology, and also about the use of machine learning in the context of ultrasound, can be found in the reflections by Noble (2016).

2.3 Structured Literature Review

An important part of this thesis was to get an overview of previous work related to the combination of neural networks, ultrasound imaging, and the prediction of image locations. To get such an overview, a structured literature review was conducted. First, key terms and groupings were searched for at several online libraries. Next, the results went through a filtering process, to extract the most relevant publications. Also, additional sources of information were found in less formal ways.

2.3.1 Literature Libraries

The following online libraries were used for the literature review:

- Springer Link
- IEEExplore
- ScienceDirect
- CiteSeer
- ACM digital library
- Web of Science

2.3.2 Key Terms and Groupings

For the library search, several key terms were selected. These terms are shown in Table 2-1, where terms with similar semantics are grouped together.

Group	Key terms
1	Deep learning Convolutional network Neural network Machine learning
2	Ultrasound Sonography Medical
3	Position Location Identify Classify

Table 2-1: Key terms and groups used in the structured literature review.

2.3.3 Search Strategy

The online libraries were searched for several Boolean expressions that provided some important combinations of the above search terms. These Boolean expressions were:

("deep learning" OR "convolutional network" OR "neural network" OR "machine learning") AND (ultrasound OR sonography OR medical) AND (position OR location OR identify OR classify)

("deep learning" OR "convolutional network" OR "neural network") AND ultrasound.

("deep learning" OR "convolutional network" OR "neural network") AND ultrasound AND location

("deep learning" OR "convolutional network" OR "neural network") AND ultrasound AND classify

2.3.4 Filtering Process

The Boolean expression searches produced too many hits – some of them several thousand – to examine all of them. To get a more manageable number of publications, only the top 15 hits of each search expression from each library were kept. Next, publications with titles that clearly indicated irrelevance were rejected.

The remaining publications were examined more closely by reading their abstracts in the light of the following inclusion criteria:

- The publication's main concern is machine learning in the context of ultrasound images.
- The publication describes a technique or an application of a technique, and presents results.

The remaining publications after this filtering were all concerned with the combination of ultrasound and machine learning, but none of them were related to identifying the origin of the acquired ultrasound data. Some of them were still partly relevant for this thesis, and these are presented in section 2.4.

2.3.5 Other Sources

In addition to the structured search described above, several other means were used to find relevant publications. This included simple library searches, examining publications used in other studies, and finding works that cited the already found publications. Still, no studies were found that were concerned with the use of neural networks to predict image locations. To the author's knowledge, no such previous studies exist.

2.4 Related Work

Several studies have been conducted on the combination of ultrasound and neural networks. However, after an extensive search, no studies were found that focused on determining the exact body location of ultrasound images by using neural networks. To the author's knowledge, no such research exists, but several studies exploring related aspects were found. These will be presented below.

2.4.1 Known Body Location, Unknown Object Location

Some of the sources that were found dealt with the determination of location, but not the location of the image itself; rather, they focused on determining the location of an object *within* the image. Both studies presented below used a classification network to accomplish this, and both studies used medical images as input to the networks.

In (Liefers et al., 2017), a neural network was used to determine the center of the fovea in Optical Coherence Tomography (OCT) scans. Although the OCT scans are normally aimed at the approximate foveal center, an automated post-processing method was suggested to determine the true foveal center more accurately, as this would reduce errors when later extracting biomarkers.

To accomplish this, a convolutional neural network was set up, taking OCT scans as input, and resulting in two classification nodes with the soft-max function applied. The network was trained on 781 OCT scans, where each pixel of the scans had been manually labelled as fovea or not. When running a new scan through this network, the network classified each pixel as fovea or not fovea, with a certain probability. The center of the fovea was then determined by selecting the pixel that had the highest probability of being a fovea pixel.

When evaluating the performance of this method, a prediction was deemed successful if the coordinate lay within the fovea region (a circle of about $1.5\ mm$ diameter). In 96.9% of the cases, the network predicted the fovea center correctly.

(Smistad & Løvstakken, 2016) used a rather different method to detect the position of blood vessels in ultrasound scans. Automatic blood vessel detection could be useful for several applications, like deep venous thrombosis detection and guiding regional anesthesia with ultrasound prior to surgery (Gray, 2006; Gupta, Gupta, Dwivedi, & Jain, 2011; Taylor, 2003). In the latter case, the awareness of blood vessels is important to avoid unintentional intravenous injection of the anesthetic medium.

The first step of Smistad's method was to detect dark ellipses in the image, to serve as blood vessel candidates, since blood vessels are often elliptically shaped in ultrasound images and blood is displayed as dark pixel due to the low echogenicity. The vessel candidate search was done by testing ellipses at many image locations and of many sizes. If the normals along the ellipse boundary corresponded well with the image gradients at the same points, that elliptic region of the image was cropped and saved as a vessel candidate. The second step was to hand the candidates to a convolutional neural network that was trained to classify images as blood vessel or not.

The network was similar to the AlexNet (Krizhevsky et al., 2012), but somewhat simplified. On average, the classification accuracy of the network was 94.5%.

2.4.2 Classification of Lesion

Classification of lesions in medical images as malignant or benign are not related to determination of image location, but are still interesting in this thesis, as they combine neural networks with medical image data. The best way to determine for certain if a lesion is malignant is to perform a biopsy, but biopsies can be painful and always introduce a risk of infection. Therefore, considerable effort is put into developing techniques that can classify lesions based only on medical image data (Alvarenga, Pereira, Infantosi, & de Azevedo, 2006; Lo & Floyd Jr, 1999).

A common way to classify lesions based on image data seems to be an initial manual extraction of image features, followed by feeding this feature vector through a neural network. Alvarenga et al. (2006), Lo and Floyd Jr (1999), and Singh, Verma, and Thoke (2016) are all examples of successful applications of this technique. However, for this thesis, techniques that use images as direct input to neural networks are more relevant.

In (Paul, Plassard, Landman, & Fabbri, 2017), images of brain tumors were fed directly to a neural network, for classification into one of three tumor types. The network consisted of two convolution/max-pooling layers, followed by two fully connected layers and a final softmax layer with 3 nodes. Several modifications were tested, like different image sizes, providing the network with the tumor location, and zooming in on the tumors, as well as experimentation with hyperparameters like learning rate, momentum, batch size and the number of epochs.

The best results were produced with the larges images sizes (256×256), no pre-processing of the images, and using only the images as input, which is a good indication that the method might be transferable to other applications. At its best, the network predicted the correct tumor type 90.26% of the time, which was better than existing methods that took more specialized approaches. This points to neural networks as a powerful general method for many domains. Interestingly, any amount of learning rate decay introduced in this study produced significantly worse accuracies.

2.4.3 Regression on Image Input

When neural networks are used for regression, the far most common input seems to be feature vectors. A typical example of this is Chiarazzo, Caggiani, Marinelli, and Ottomanelli (2014), where a neural network is used to estimate real estate prices. In this study, the input of each example was a vector with over 30 features, describing properties of a real estate, e.g. if the estate was rural, if it had a garden, how much air pollution was present, etc. The output of the network was a single node, which estimated the market value of the real estate.

Although feature vectors are the most common input, images can also easily be fed to a regression network. In that case, the network may be very similar to a CNN used for classification, except the nodes of the output layer are not transformed with a soft-max function. Instead, the values of each node are read out independently to represent some property. An example of using a CNN for predicting the angle of rotation on handwritten digits is made available by MathWorks (The MathWorks, 2017). Here, the input is an image, and the output is a single node, representing the angle of rotation.

As an example of using regression on images for medical purposes, a study by (Xue, Ray, Hugh, & Bigras, 2016) is briefly presented. The motivation for this study was the problem of counting the number of cells in an image, which can be important for precision diagnostics in laboratory medicine. For example, the number of tumor cells may be a good indicator of the severity of the cancer. An automated counting technique is highly desirable, as manual counting is very time-consuming, or even infeasible.

In the training phase of this technique, the training images were cut into many smaller patches, and the label of each patch was the number of cells in each patch, counted manually. Next, the patches were augmented by rotation, to increase the training set, and to provide invariance to image rotation. Finally, two CNNs, one of them similar to the AlexNet, were trained on a large number of such patches, and learned to output a single number, indicating the cell count.

In the test phase, a test image was cropped into many patches of the same size as the training patches. The patches were cropped using a sliding window that caused a lot of overlapping. Next, the regression network produced an estimate of the cell count for each patch. These cell counts were then interpolated in 2 dimensions, to produce a heatmap that hopefully averaged

out some of the prediction errors. In the end, the heatmap was integrated to produce the final cell count.

The counting performance was tested on three different datasets. The error of the AlexNet-like network on these datasets ranged between 4.5% and 16.7% of the correct cell count, which was significantly better than several recent related methods.

2.4.4 Unknown Body Location

Only a few studies were found that were related to finding certain body locations, and none of these used the technique applied in this thesis. In Kerby, Rohling, Nair, and Abolmaesumi (2008), an attempt was made at automatically labelling the lumbar vertebrae, although not by using machine learning. The lumbar vertebrae are the five vertebrae located between the pelvis and the rib cage, and when performing an epidural anesthesia, the correct localization of these vertebrae is important, as the needle should preferably be inserted between the third and the fourth lumbar vertebrae (L3 and L4). The traditional identification method is manual palpation, but this method is known to be imprecise, so the authors designed and tested a new automatized labelling method.

The method consisted of placing an ultrasound probe near the coccyx (the tailbone) in a parasagittal plane. The probe was then moved slowly upwards along the spine while recording, producing a series of overlapping ultrasound images. These images were stitched together to form a panorama image of all the lumbar vertebrae, the sacrum, and the coccyx. Next, an image processing algorithm was applied to detect the vertebrae in the panorama image. This algorithm searched the image for bone structure peaks, which, in ultrasound images, are typically rendered as bright echoes followed by dark regions. During the stitching process, the probe movement was estimated by measuring the movement of features from image to image. This, combined with the relatively simple palpation of the coccyx, allowed for drawing the position of the lumbar vertebrae on the skin.

The method was tested on ten patients, and the locations were compared to a sonographer's estimation of the correct locations, using a freehand ultrasound scan and palpation. The method successfully labelled all the vertebrae in all ten trials, but the exact midpoint of the vertebrae differed from the sonographer's estimations by 6.5 mm on average.

(Yu & Tan, 2014) also attempted to simplify the identification of the lumbar vertebrae, by using machine learning. In this study, the focus was not on automatically identifying the vertebrae levels, but rather to distinguish between bone and inter-spinous ultrasound images taken from the lower spine in the transverse plane. By giving such feedback in real-time, the probe operator could more easily find the desired vertebrae by counting from the sacrum and upwards.

An artificial neural network was set up to classify the images. Instead of using raw images as input, the network was given a vector of features, extracted from the ultrasound images by image processing techniques. For example, the midline of the image was calculated, and the appearance of dark pixels along this midline was extracted as one of the features, since this was an indication of sonic shadow from bone. Another extracted feature was the appearance and position of a 'flying bat' shape (found by template matching), which is typical for inter-spinous images.

The network was trained on 1000 images from 25 patients, and tested on 720 images from 18 other patients. More than 94% of the images were classified correctly, and the study concluded that the method could facilitate the anesthetist 's identification of needle insertion positions.

Another interesting attempt at automatically finding the correct body location, is the product prototype Veebot (LLC, 2014), a device that automatically detects the best spot for venipuncture blood draws on the lower arm, and then performs the procedure. No papers have been published by the company yet, but a profile of the product has been published by the IEEE Spectrum (Perry, 2013).

In the beginning of the process, a patient places his/her lower arm on the Veebot device, in such a way that movement is restricted. Next, the device illuminates the lower arm with infrared light, to highlight the blood veins for a camera. Using image analysis, the acquired images are template matched against a vein model, built from a collection of thousands of images of suitable veins, in order to select the best candidate for venipuncture. Next, a Doppler ultrasound device is used to confirm that the blood flow in the selected vein is sufficient. When the vein is confirmed, a robot arm steers the needle towards the vein, and aligns it with the vein's orientation, before finally puncturing the vein and drawing blood. The whole procedure takes about one minute, and the company claims that Veebot selects the best vein to target 83% of the time – about the same level as an experienced technician.

2.4.5 Determining Image Source from Raw Image Input

This final category of the related works section discusses the problem of determining the source of an image (where it is taken, or what it depicts), using only the raw image as input to a machine learning system – i.e. without first extracting any features from the image using image analysis tools. The current dominant branch of machine learning used to accomplish this is artificial neural networks, and more specifically, CNNs.

In the 1990's, Yann LeCun developed the first successful CNN (LeCun, Bottou, Bengio, & Haffner, 1998). In this network, the pattern of convolution/pooling/non-linearity sequences were introduced, and the non-linearity function was then implemented either as a hyperbolic tangent or as a sigmoid. The last part of the network consisted of fully-connected layers that ended in a 10-node output classification layer. Earlier networks by LeCun (LeCun et al., 1989) had been utilized to automatically interpret hand written zip codes with raw images as input, but the new convolutional layers took advantage of the inherit importance of spatial proximity in images, reducing the number of parameters in the network and thus allowing for larger networks.

Between 1998 and 2010, CNN's received little attention, but in 2010, the annual ImageNet Large Scale Visual Recognition Challenge was initiated (Stanford Vision Lab, 2016), challenging participants to design systems capable of classifying images with 1000 possible categories. For training these systems, millions of images (or image URLs) are available in the ImageNet database. In 2012, the winner of the challenge was a CNN by Alex Krizhevsky called AlexNet (Krizhevsky et al., 2012), and after this, deep convolutional neural networks (hence the name deep learning) was regarded as the best available machine learning technique for classification of images with raw images as input. One of the contributions of the AlexNet was the implementation of rectified linear units (Nair & Hinton, 2010) as the non-linearity functions, replacing the sigmoid and tanh functions. Another important improvement was the use of the dropout technique (Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 2012) to greatly reduce overfitting. An implementation of AlexNet for Keras is included in appendix 9.2.

After this, many improved CNNs were designed and they continued to improve the score on the ImageNet competition. In 2014, a series of networks called VGG (Simonyan & Zisserman, 2015) won the ImageNet competition. The main contribution of these networks was a reduction

of the convolution filter size (only 3×3), together with an increase in the depth of the network – up to 16 and 19 layers deep. In contrast, the largest convolution filter size of the AlexNet was 11×11. The key insight was that the detection of large image features could be accomplished by stacking many smaller convolutions after each other, which at the same time increased the number of abstraction levels in the model. An implementation of the 16 layer VGG network (VGG-16) for Keras is included in appendix 9.3.

In a previous study by the author (Fossan, 2016), CNNs were trained to classify ultrasound images as originating from one of three locations on the body. As in the ImageNet competition, only raw images were used as input to the networks. The image set was acquired by making ultrasound recordings of three typical locations for ultrasound-guided placement of anesthetic nerve blocks: On the arm, just above the wrist; On the axilla, for placement of the axillary block, and; On the side of the neck, for placement of the interscalene block. The recordings, ranging from 30 seconds to two minutes, were done on four persons, with two recordings per person per location.

The network used in this study was a custom-made CNN, consisting of two sequences of convolution/ReLU/Max-pool, one fully connected layer with dropout, and a final fully connected classification layer with three nodes. The convolution filter sizes were 5×5, the max-pool filter sizes were 2×2, and the first fully connected layer consisted of 512 nodes.

Because of the small number of persons involved, the cross-validation average classification accuracy was measured. The accuracy of the network was about 74.4%. The study was a first attempt at using neural networks to guide an ultrasound operator towards a certain body location. Although the classification of ultrasound images into one of three possible locations is not of imminent practical use, the results showed that CNNs have the potential of classifying the source of ultrasound images. A first step of guiding an ultrasound operator could be to determine approximately where on the body the ultrasound probe is positioned, for example by using a classification network to determine which body part the probe is looking at (arm, leg, neck, etc.). This could then be followed by a more precise localization technique.

2.5 Baseline for Results

All the experiments performed in this thesis are concerned with one of the following metrics:

1. Classification accuracy
2. Error of regression along one dimension
3. Error of regression along two dimensions

In order to have a worst-case baseline to compare the results against, this section derives the expected values of the above metrics in the case when all predictions are done randomly. To check the validity of the derived formulas, simulations of the random number selections were conducted. The results of these simulations where in agreement with the formulas.

2.5.1 Classification Accuracy

The expected classification accuracy with random predictions is straight forward. With m classes to choose between,

$$E\left(accuracy_{random\_classification}\right) = \frac{1}{m}$$

For example, in the first group of experiments of this thesis (regional anesthesia experiments, section 3.1), there are three classes to choose between, so the expected random classification accuracy is $\frac{1}{3}$.

2.5.2 Error of Regression Along One Dimension

In this case, the task is to predict a number along a single dimension. The derivation below assumes that both the correct position and the predicted position are chosen randomly along a line of length L. The source used for these calculations was Carvalho (2013).

Let X be a random variable evenly distributed over the allowable range $[0, L]$. The probability density function of X will then be

$$pdf_X(x) = \frac{1}{L}$$

, since the integrated sum of pdf_x over the allowable range is 1

$$\int_0^L \frac{1}{L} dx = 1$$

Further, let X_1 and X_2 be the predicted position and the correct position, respectively, selected randomly inside $[0, L]$ and distributed according to pdf_X.

We are trying to find the expected distance between X_1 and X_2, so let Y be a new random variable representing this distance, so that $Y = |X_1 - X_2|$.

Further, let $g(x_1, x_2) = |x_1 - x_2| = \begin{cases} x_1 - x_2 & if\ x_1 \geq x_2 \\ x_2 - x_1 & if\ x_1 < x_2 \end{cases}$

The joint probability density function of X_1 and X_2 is the product of their respective probability density functions, since they are chosen independently, so

$$pdf_{X_1X_2}(x_1, x_2) = pdf_{X_1}(x_1) pdf_{X_2}(x_2) = \frac{1}{L^2}$$

, inside the allowable range.

Y, the expected distance between X_1 and X_2, is the sum of the distances between X_1 and X_2 for all possible choices of X_1 and X_2, times the probability of each such pair being chosen.

$$\begin{aligned} E(Y) &= \int_0^L \int_0^L g(x_1, x_2) pdf_{X_1X_2}(x_1, x_2) dx_2 dx_1 \\ &= \frac{1}{L^2} \int_0^L \int_0^L |x_1 - x_2| dx_2 dx_1 \\ &= \frac{1}{L^2} \int_0^L \int_0^{x_1} (x_1 - x_2) dx_2 dx_1 + \frac{1}{L^2} \int_0^L \int_{x_1}^L (x_2 - x_1) dx_2 dx_1 \end{aligned}$$

In the last line, x_2 is first integrated up to the position of x_1 using the function $(x_1 - x_2)$. Next, x_2 is integrated up to L, but with the formula $(x_2 - x_1)$, since x_2 is now greater than x_1.

Continuing,

$$
\begin{aligned}
E(Y) &= \frac{1}{L^2}\int_0^L \left(x_1^2 - \frac{1}{2}x_1^2\right)dx_1 + \frac{1}{L^2}\int_0^L \left(\frac{1}{2}L^2 - Lx_1 - \frac{1}{2}x_1^2 + x_1^2\right)dx_1 \\
&= \frac{1}{L^2}\left(\frac{1}{6}x_1^3\Big|_0^L\right) + \frac{1}{L^2}\left(\frac{1}{2}L^2x_1 - \frac{L}{2}x_1^2 + \frac{1}{6}x_1^3\Big|_0^L\right) \\
&= \frac{1}{L^2}\left(\frac{L^3}{6}\right) + \frac{1}{L^2}\left(\frac{L^3}{6}\right) \\
&= \frac{L}{6} + \frac{L}{6} = \frac{L}{3}
\end{aligned}
$$

In the experiments of this thesis, the allowable range for the predicted and correct numbers is [0,1], so $L = 1$. The expected distance with random predictions is therefore $\frac{1}{3}$.

2.5.3 Error of Regression Along Two Dimensions

In this case, the task is to predict a position inside a rectangle. In Burgstaller and Pillichshammer (2009), the expected distance Y between two randomly selected points inside a rectangle of sides a and b, where $a \geq b$, is given by

$$
E(Y) = \frac{1}{15}\left[\frac{a^3}{b^2} + \frac{b^3}{a^2} + d\left(3 - \frac{a^2}{b^2} - \frac{b^2}{a^2}\right) + \frac{5}{2}\left(\frac{b^2}{a}\log\frac{a+d}{b} + \frac{a^2}{b}\log\frac{b+d}{a}\right)\right]
$$

, where $d = \sqrt{a^2 + b^2}$

In this thesis, two different values for a and b are considered:

1. $a = b = 1$

Then $d = \sqrt{2}$ and the expected distance between the two points is

$$
\begin{aligned}
E(Y) &= \frac{1}{15}\left[2 + \sqrt{2} + 5\left(\log\frac{1+\sqrt{2}}{1}\right)\right] \\
&\approx 0.521
\end{aligned}
$$

2. $a = 1$ and $b = 2$

Then $d = \sqrt{5}$ and the expected distance between the two points is

$$E(Y) = \frac{1}{15}\left[\frac{1}{4} + 8 + \sqrt{5}\left(3 - \frac{1}{4} - 4\right) + \frac{5}{2}\left(4\log\frac{1+\sqrt{5}}{2} + \frac{1}{2}\log\frac{2+\sqrt{5}}{1}\right)\right]$$
$$\approx 0.805$$

3 Methods

This chapter presents the methods used to investigate the goals and research questions of this thesis. Five datasets were used, and several experiments were conducted for each dataset. For readability, separate sections are given to each dataset in this chapter. The last section describes the technical set up common to all the experiments.

3.1 Regional Anesthesia Dataset: Rough Classification of Image Origins

This section describes experiments concerning the classification of the approximate origin of ultrasound images. The experiments were performed on the "Regional Anesthesia Dataset" – the same dataset used in Fossan (2016). The main purpose of the experiments was to test the effect of pre-trained neural networks in the context of ultrasound images, as a preparation for the experiments on the "Lower Arm Dataset" in section 3.4. The experiments were a continuation of the work done in Fossan (2016), as described in section 2.4.5, and the effect of pre-training was tested by repeating the classification task of that study with and without pre-training applied (experiments 1 and 2), using a person-wise cross-validation scheme.

A third experiment was conducted where the network trained on 90% of the images from all persons, instead of using a cross-validation scheme. In the final experiment, one of the test persons – the only female – was removed. Using this person as the test set produced very poor results in the previous study, and this person also scored the lowest on the first two experiments in the current thesis. The purpose of this final experiment was to see if the cross-validation accuracy of the remaining persons would be improved by the removal.

3.1.1 Description of Dataset

The dataset consisted of ultrasound images acquired from four test persons. For each person, recordings were done on three different body locations, all typical locations for ultrasound-guided placement of anesthetic nerve blocks (Gray, 2006; TorontoWesternHospital, 2017).

These locations were

a) On the arm, just above the wrist
b) On the axilla, on the inside of the upper part of the arm, near the shoulder joint – for placement of the axillary block.
c) On the side of the neck, near the collarbone – for placement of the interscalene block.

Figure 3.1 shows examples of images from the three locations.

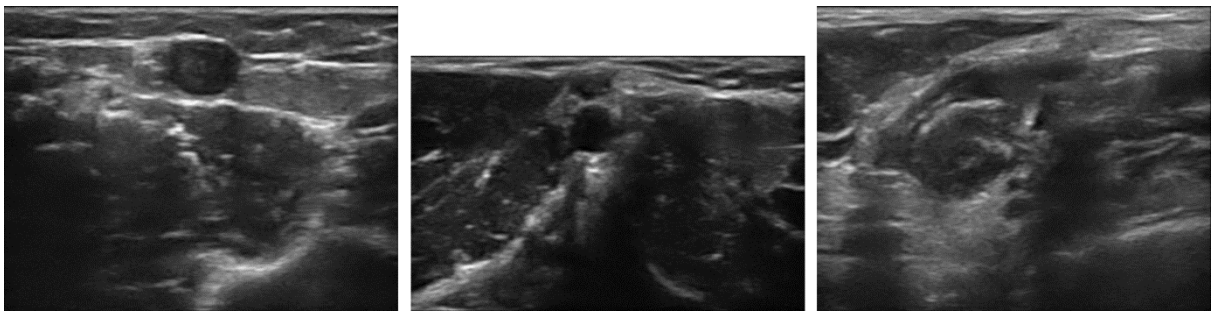

Figure 3.1: Examples of ultrasound images in the regional anesthesia dataset.(**Left**) Image of the lower arm. (**Center**) Image of the Axilla. (**Right**) Image of the side of the neck.

For each person, two or three ultrasound recordings were made per body part, each ranging between 30 seconds and 2 minutes. In total, this produced 18 439 images. Each image was given a one-hot encoded label describing which location that image had been acquired from.

3.1.2 Description of Network

In order to apply pre-training, the neural network used in the previous study had to be replaced. Since the neural network in that study was custom made, no pre-trained versions of it existed. Instead, the vgg16 network was chosen (see section 2.4.5), as pre-trained versions of it were available through Keras. The implementation of VGG16 for Keras can be found in appendix 9.3.

The structure of layers 1 through 5 were kept intact, to allow the pre-trained weights to be copied. Each of these five layers consisted of convolution – activation – max pooling sub layers, used for feature extraction. However, the final layers were modified to make the network fit with the classification problem, as shown below. For an explanation of the Keras code, see appendix 9.1.

```
# Layer 6
Flatten()
Dense(512)
Activation('relu')
Dropout(0.5)

# Layer 7
Dense(3)
Activation('softmax')
```

Two changes were made in these layers. First, the number of nodes in layer 6 were reduced to 512, and the original layer 7 was dropped. This was done to speed up training time. Second, the number of nodes in the output layer (now layer 7) was reduced to three, since the task was to distinguish between three classes.

3.1.3 Training and Testing

In these experiments, the training set consisted of images from three of the persons, while the images from the fourth person were assigned to the test set. 100 images were used from each recording to avoid overrepresentation of some of the locations. After training on the three persons in the training set, the network was given the task of classifying the unseen images in the test set. This procedure was repeated using cross-validation, so that the persons took turns acting as the test set. Finally, the average classification performance was calculated.

The loss function used during training was the cross-entropy, as described in section 2.1.3.1 (Eq. 2-1). The function is repeated below for convenience.

$$-\frac{1}{N}\sum_{i}\sum_{j=1}^{3} y_{ij}\log(prediction_{ij})$$

Here, y_i is the correct label, and $prediction_i$ is the predicted label of the i^{th} ultrasound example of the current batch. Since there are three possible categories, both $y_i$and $prediction_i$ are vectors of size three. The scores of all examples in the batch are summed and divided by the batch size, N, to give the average loss value of the batch. The Adam optimizer was used to perform the minimization of this loss function.

During testing, the network's ability to correctly classify unseen images was measured. This accuracy was measured by comparing the network's softmax output layer to the correct labels for each image. In the output layer vector, the element with the highest value signaled the

predicted image location, and when comparing the *argmax* of this vector to the *argmax* of the label vector, a *one* would indicate a match, while a *zero* would indicate a mismatch. Doing this over the entire test set and dividing by n gave the overall classification accuracy in percentage. The formal function is given below.

$$\frac{1}{n}\sum_{i} equals(argmax[prediction_i], argmax[y_i]) \tag{3-1}$$

The accuracy on the entire test set was calculated after each epoch, to see the development of the accuracy over time. Also, each experiment (training and testing) was run five times, to measure the average performance.

The same modified VGG16 network structure was used in all four experiments described below. In exp. 2 and 3, were pre-training was applied, layers 1 through 5 were loaded with pre-trained weights. Also, these layers were frozen, meaning they were not modified during the following training process. Instead, further training took place only in the last fully connected layers.

3.1.4 Experiment Descriptions

3.1.4.1 Experiment 1 (cross-validation, no pre-training)

Property	Value
Total number of images	2400
Training images	1800 (images from three persons)
Test images	600 (images from one person, cross-validation)
Pre-training	No
Runs	5
Epochs	10
Batch size	10
Loss function	Cross-entropy (Eq. 2-1)
Test function	Accuracy (Eq. 3-1)
Optimizer	Adam optimizer
Learning rate	10^{-5}

Table 3-1: Properties of regional anesthesia experiment 1.

3.1.4.2 Experiment 2 (pre-training applied)

Similar to exp. 1, except that pre-training was applied. Layers 1 to 5 were loaded with pre-trained weights, and frozen for further training.

3.1.4.3 Experiment 3 (train and test on all persons)

This experiment also used pre-training, and was similar to exp. 2, except for the partitioning of the images into the training set and the test set. This time, 90% of the 100 images extracted from each recording were assigned to the training set, while the remaining 10% where assigned to the test set, so that the network was trained and tested on images from all persons. The differences from exp. 2 are summarized in Table 3-2.

Property	Value
Training images	2160 (90% of the images from all recordings)
Test images	240 (10% of the images from all recordings) Cross validation not applicable

Table 3-2: Differences between regional anesthesia experiments 2 and 3.

3.1.4.4 Experiment 4 (remove person 3)

This experiment was based on exp. 1 (cross-validation, no pre-training). The only difference was the removal of the third person in the dataset from the training and testing. The differences, compared to exp. 1, are shown below:

Property	Value
Total number of images	1800
Training images	1200 (images from two persons instead of one)

Table 3-3: Differences between regional anesthesia experiments 1 and 4.

3.2 1D Synthetic Ultrasound Dataset: Regression in One Dimension

These experiments focused on testing the concept of training neural networks with regression. This meant that, in contrast to the regional anesthesia experiments, where image pixel intensities where mapped onto three classification variables, the goal of the experiments in this section was to map image pixels onto a single variable taking *continuous* values.

The dataset images were synthetically created to simulate a single one-directional ultrasound scan of a volume, and the continuous output variable of the network would in this case represent the location of the ultrasound probe for each image. The image labels were numbers between 0 and 1, where 0 represented the start location of the scan, and 1 represented the end location. By

training on some of the images in the dataset, the network should learn to predict the location of the other, unseen images.

Also included was an experiment that introduced simple data augmentation, to see if it could have a positive effect on the final accuracy.

3.2.1 Description of Dataset

The dataset used in these experiments consisted of 37 synthetic images of size 100×64, designed to simulate a one-directional ultrasound scan of a volume. The imaginary volume, shown in Figure 3.2, contained a cone and a plane, and the ultrasound probe can be imagined placed on top of this volume, looking down into it. The scan was performed front-to-back, starting with frame0 and ending with frame36, as shown in the figure.

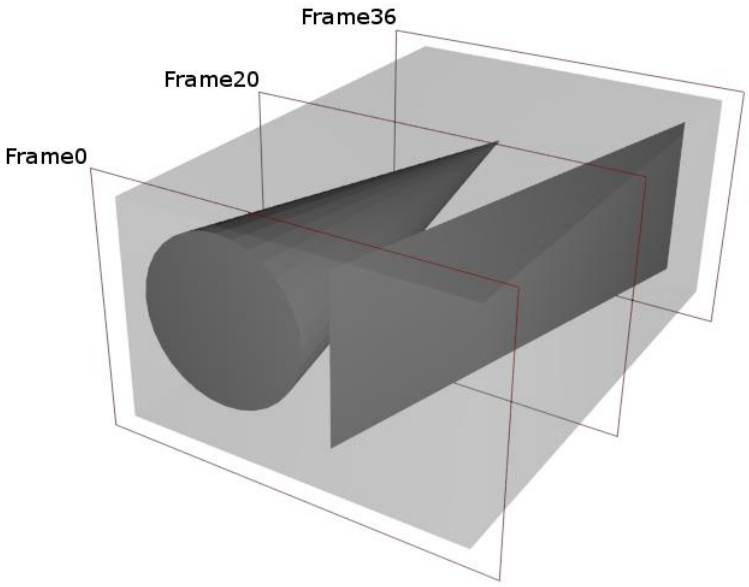

Figure 3.2: Visualization of the imaginary volume that the 1D synthetic ultrasound dataset was based on. The volume consisted of a cone and a plane, and the recording was done from front to back.

Three examples of the synthetic images are shown in Figure 3.3. As can be seen, some noise was added to make the images look more like ultrasound images. This noise was produced by first adding random patches of white and black, and then performing radial motion blur on these patches, with the blur rotation center placed at the top center of the image, to mimic increasingly blurred images further away from the ultrasound probe. The purpose of adding noise was to

make the regression problem more realistic, since inferring image location from clean shapes without noise would presumably be simple.

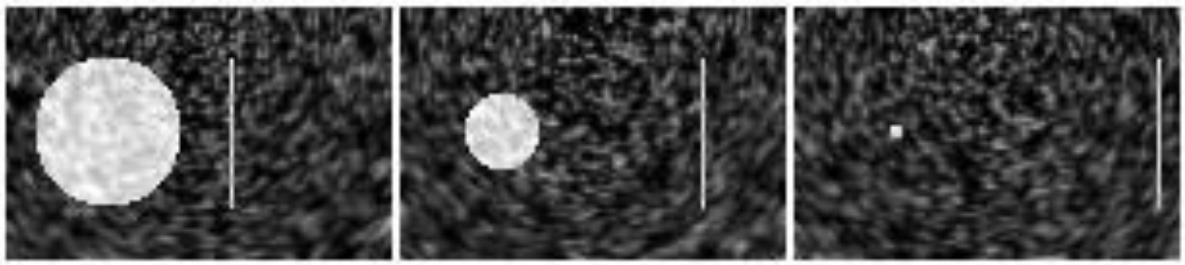

Figure 3.3: Examples of images in the 1D synthetic ultrasound dataset. (**Left**) The first frame in the recording sequence, Frame0, where the cross section of the cone is large, and the cross section of the plane is close to the center. (**Center**) Frame20. (**Right**) The last frame, Frame36, where the cross section of the cone is small, and the cross section of the plane has moved to the right side of the image.

In the first frame of Figure 3.3 the cone is visualized as a large disk, and the plane appears as a vertical line close to the center of the image. For each consequent frame, the disk diminishes and the line moves towards the right edge of the image. This was the pattern that the neural network would have to learn in order to predict image locations.

3.2.2 Description of Network

The neural network used in these experiments was based on AlexNet (Krizhevsky et al., 2012) – more specifically it was based on a Keras-implementation of AlexNet (Yale University, 2016), designed for use on the MNIST dataset. Although not identical to the original AlexNet, this implementation captures the main structure of the original layout. It consists of eight layers, where each layer contains several sub-functions. The Keras-implementation can be found in appendix 9.2.

For these experiments, the AlexNet implementation was modified somewhat. The complete resulting network is shown below.

```
# Layer 1
Convolution2D(32, 5, 5, input_shape = (64,100,1))
Activation('relu')
MaxPooling2D(pool_size=(2, 2))

# Layer 2
Convolution2D(64, 5, 5)
Activation('relu')
MaxPooling2D(pool_size=(2, 2))

# Layer 3
ZeroPadding2D((1,1))
Convolution2D(128, 3, 3)
Activation('relu')

# Layer 4
ZeroPadding2D((1,1))
Convolution2D(256, 3, 3)
Activation('relu')

# Layer 5
Convolution2D(512, 3, 3)
Activation('relu')
MaxPooling2D(pool_size=(2, 2))

# Layer 6
Flatten()
Dense(1024)
Activation('relu')
Dropout(0.5)

# Layer 7
Dense(512)
Activation('relu')
Dropout(0.5)

# Layer 8
Dense(1)
```

Two modifications were made to produce this network. First, in order to make it fit into the different GPUs that were utilized, the network was simplified by reducing the number of parallel convolutions, by reducing the convolution kernel sizes and by reducing the fully connected layers. Second, the last layer was reduced to only include one output node with continuous values, enabling this network to handle a regression problem instead of a classification problem. This single output node represented the predicted location of the ultrasound probe for each image.

3.2.3 Training and Testing

For these simple experiments, the group of images was split in two; Every second image was assigned to the training set, while the rest were assigned to the test set.

While the cross entropy was a suitable loss function for the classification problem in section 3.1, a more natural choice in the context of regression was to use the mean squared error (MSE), as described in section 2.1.3.1 (equation 2-2), and repeated below.

$$\frac{1}{N}\sum_{i}(y_i - prediction_i)^2$$

However, a different function was applied during testing; the mean absolute error (MAE), described in section 2.1.3.1 (equation 2-4), and repeated below.

$$\frac{1}{N}\sum_{i}|y_i - prediction_i|$$

The reason for using MAE instead of MSE during testing, was to make it easier to relate the output to the physical context of the experiments. Having the same scale and unit as the labels, the MAE could be directly interpreted as the average distance between the predicted image locations and the real image locations. For example, a MAE of 0.1 would mean that, on average, the network's predicted locations were wrong by 0.1 units, which corresponds to 10% of the total length of the recording, since the images in the recording were labelled from 0 (start) to 1 (end). At the same time, MSE was preferred over MAE during training, since the square in MSE makes large errors have relatively greater influence on the loss than small errors.

After each epoch of training, the network was tested on the whole test set, to get an exact view of the MAE development over the epochs.

An experiment was also performed to test the effect of simple data augmentation in the context of regression training. The data augmentation was designed to simulate two factors, the first being random displacements of the ultrasound probe in the sideways direction, and the second being that the structures inside the volume could have varying distances from the recording surface. These factors were simulated by randomly shifting images in the x- and y-direction by

up to 10% of the image width and height. This kind of data augmentation is a built-in feature of Keras, and was easy to apply in the experiment.

In order to measure the average results over multiple runs, each experiment described below was run five times.

3.2.4 Experiment Descriptions

3.2.4.1 Experiment 1 (no data-augmentation)

Property	Value
Total number of images	37
Training images	19 (every second image)
Test images	18
Data augmentation	No
Runs	5
Epochs	100
Batch size	10
Loss function	Mean squared error (Eq. 2-2)
Test function	Mean absolute error (Eq. 2-4)
Optimizer	Adam optimizer
Learning rate	10^{-4}

Table 3-4: Properties of 1D synthetic ultrasound experiment 1.

3.2.4.2 Experiment 2 (data augmentation)

Same as exp. 1, except that data augmentation was applied, with a width- and height shift of maximum 10%.

3.3 2D Synthetic Ultrasound Dataset: Regression in Two Dimensions

In the experiments of this section, a second dimension was added to the regression problem. Now, instead of only predicting the ultrasound probe's location in the longitudinal direction, the network would also learn to predict the sideways offset of the probe. The output of the networks was therefore two variables taking continuous values – one for each dimension.

Also, several changes to the learning process were tested, to find a good balance between training time and precision. These tests were done as a preparation for the experiments on the "Lower Arm Dataset" in section 3.4, which would use real ultrasound data, and included changes to the network topology as well as the hyperparameters *learning rate*, *decay*, *batch size*, and *l2-regularization*. Since it was assumed that the size of the lower arm dataset in section 3.4 would be far greater, the testing of different hyperparameter values would be conducted much quicker in the experiments of this section. Section 3.3.4 describes some of the different experiments that were performed to test these changes. A few more experiments were performed as well, but they were not included in this thesis, as the results were less interesting.

3.3.1 Description of Dataset

The dataset used in these experiments consisted of 684 synthetic images of size 100×64, designed to simulate several one-directional ultrasound scans of a volume that represented the lower part of the arm. The scans were performed with varying sideways offset, and, to simulate recordings on different people, with varying internal structures of the volume. The imaginary volume, shown in Figure 3.4, contained two tubes that were separate in one end and merged in the other end of the volume. These tubes represented structures that change as the ultrasound probe is moved across a volume, like blood veins. The ultrasound probe could be imagined placed on top of this volume, looking down into it, and moving from the front of the volume (frame0) towards the back (frame18).

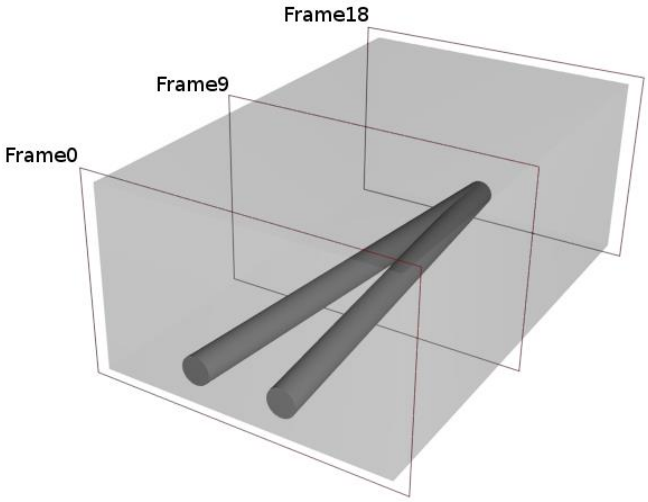

Figure 3.4: Visualization of the imaginary volume that the 2D synthetic ultrasound dataset was based on. The volume consisted of two tubes that were separate in one end of the volume, and merged in the other end.

Figure 3.5 shows the synthetic ultrasound images corresponding to the setup in Figure 3.4. Here, the tubes are visualized as bright disks that move towards each other and finally merge in the last frame. This pattern would have to be learned by the network to determine the y-coordinate.

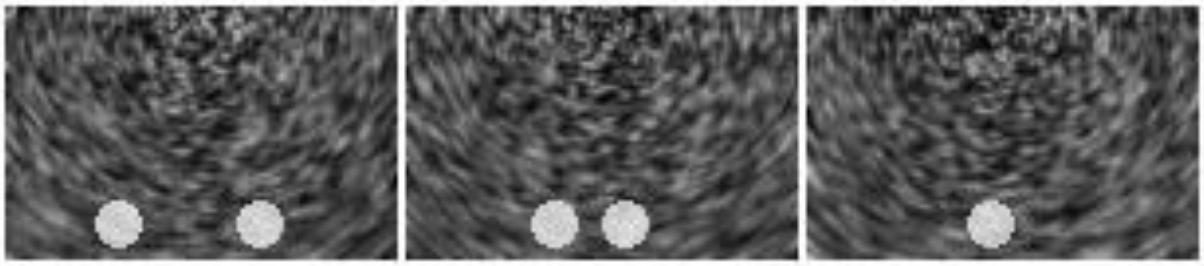

Figure 3.5: Examples of images in the 2D synthetic ultrasound dataset. (**Left**) The first frame in one of the recordings, Frame0, where the tubes are separate. (**Center**) Frame9. (**Right**) The last frame, Frame18, where the tubes are merged.

To add a second dimension to the probe location, several scans with varying sideways offset were performed. In total, the volume was scanned at nine different offsets, including the center position. The center position and the two extremes are shown in Figure 3.6.

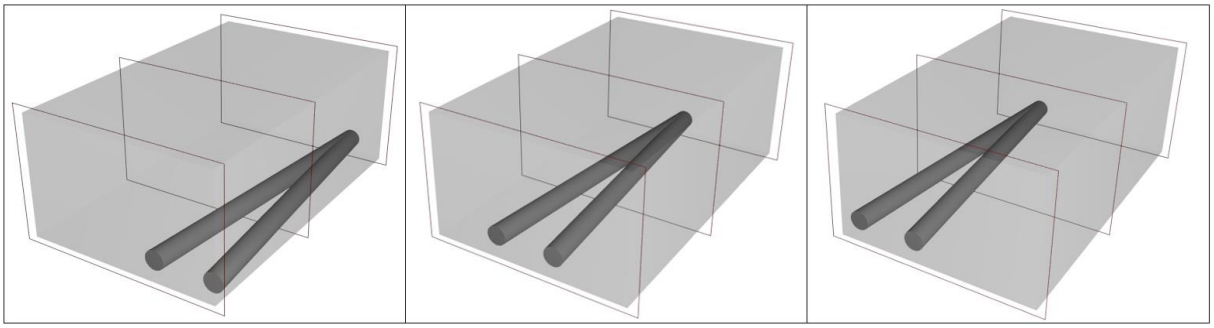

Figure 3.6: Visualization of three of the lateral offsets used in the 2D synthetic ultrasound dataset. (**Left**) Ultrasound probe placed far to the left, causing the tubes to appear far to the right in the images. (**Center**) The center position of the probe. (**Right**) Probe placed far to the right.

Figure 3.7 shows the synthetic ultrasound images corresponding to the setup in Figure 3.6.

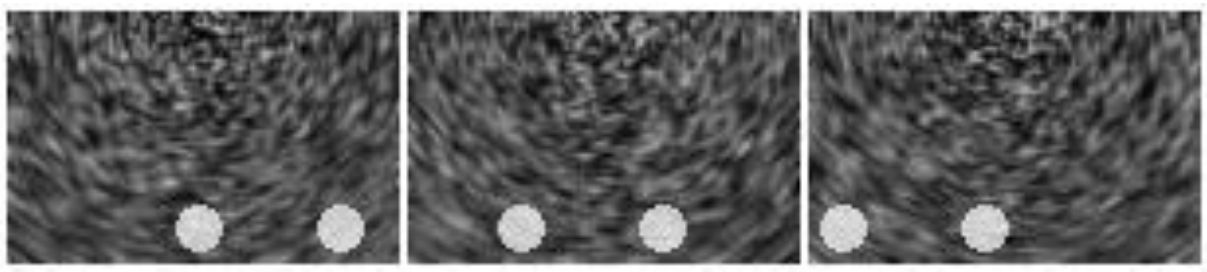

Figure 3.7: Images in the 2D synthetic ultrasound dataset with varying lateral offsets. The images correspond to the three offsets in Figure 3.6

Additionally, to simulate scans from four different people with dissimilar anatomy, the tube structure was placed at four different depths in the volume, and separate scans were performed for each depth. The four depths are visualized in Figure 3.8, and synthetic ultrasound images corresponding to these depths are shown in Figure 3.9.

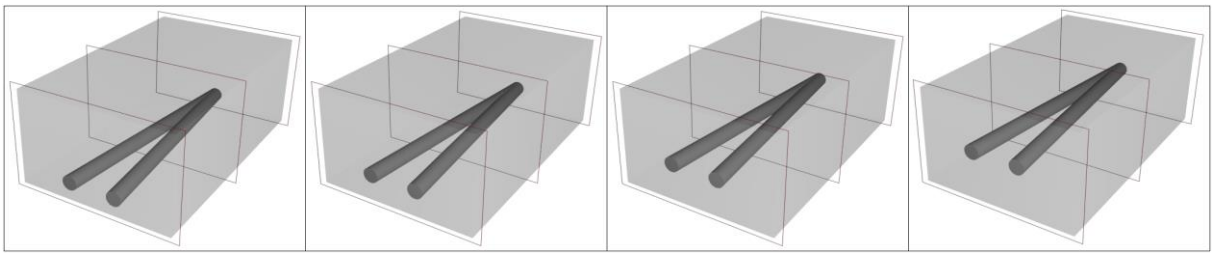

Figure 3.8: Visualization of the four tube depths used in the 2D synthetic ultrasound dataset.

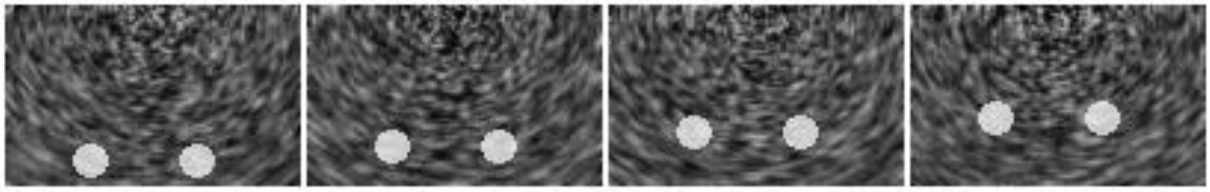

Figure 3.9: Images in the 2D synthetic ultrasound dataset showing varying depths of the tubes. The images correspond to the visualizations in Figure 3.8.

In total, 19 images per scans, nine offsets, and four depths, resulted in 684 images. Noise, similar to that of the one-dimensional synthetic dataset, was applied to the images to make them look more like ultrasound images, as can be seen in the above figures.

Figure 3.10 shows the volume seen from above with a coordinate system. This coordinate system was used when labelling the images, measured at the center of the probe. The applied parts of both axis were normalized to the range $[0, 1]$ Thus, in a scan with no horizontal offset (x-axis), the first frame would be labelled $(x, y) = (0.5, 0)$, and the last frame would be labelled $(x, y) = (0.5, 1)$. Similarly, in a scan with maximum horizontal offset to the left, the first frame would be labelled $(x, y) = (0, 0)$, and the last frame would be labelled $(x, y) = (0, 1)$.

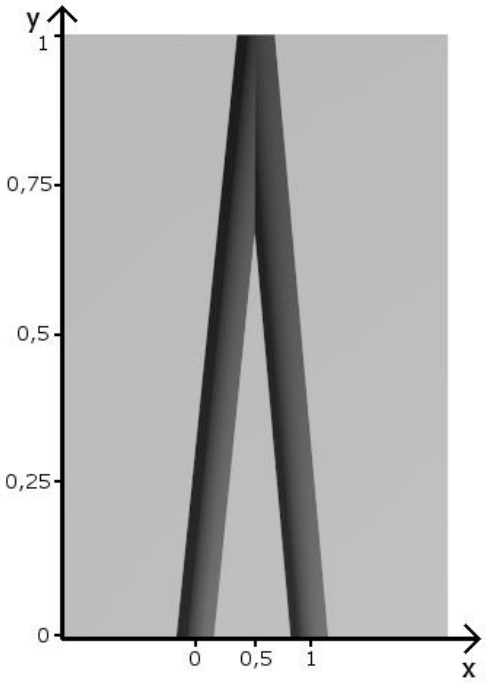

Figure 3.10: Top-down rendering of the 2D synthetic ultrasound volume, with a coordinate system.

3.3.2 Description of Networks

Several networks were used during these experiments, to see the effect of the number of layers, convolution kernel size, etc. These will be presented below.

3.3.2.1 Network 1

This network was identical to the network used in Fossan (2016), except that the output layer, consisting of three nodes and a softmax function, was replaced by two nodes with no softmax function. The whole network is shown below.

```
# Layer 1
Convolution2D(32, 5, 5, input_shape = (64,100,1))
Activation('relu')
MaxPooling2D(pool_size=(2, 2))

# Layer 2
Convolution2D(64, 5, 5)
Activation('relu')
MaxPooling2D(pool_size=(2, 2))

# Layer 3
Flatten()
Dense(512)
Activation('relu')
Dropout(0.5)

# Layer 4
Dense(2)
```

3.3.2.2 Network 2

This network was almost identical to the original Keras-implementation of AlexNet, shown in appendix 9.2. At this point, graphics cards with more memory were available, so the network was kept mostly intact. The only changes from the network shown in appendix 9.2, were to reduce the sizes of the last fully-connected layers. Also, the final layer was changed to contain two output nodes without the softmax function. Network 2 is shown below.

```
# Layer 1
Convolution2D(96, 11, 11, input_shape = (28,28,1))
Activation('relu')
MaxPooling2D(pool_size=(2, 2))

# Layer 2
Convolution2D(256, 5, 5)
Activation('relu')
MaxPooling2D(pool_size=(2, 2))

# Layer 3
ZeroPadding2D((1,1))
Convolution2D(512, 3, 3)
Activation('relu')

# Layer 4
ZeroPadding2D((1,1))
Convolution2D(1024, 3, 3)
Activation('relu')

# Layer 5
ZeroPadding2D((1,1))
Convolution2D(1024, 3, 3)
Activation('relu')
MaxPooling2D(pool_size=(2, 2))

# Layer 6
Flatten()
Dense(1024)
Activation('relu')
Dropout(0.5)

# Layer 7
Dense(1024)
Activation('relu')
Dropout(0.5)

# Layer 8
Dense(2)
```

3.3.2.3 Network 3

This network was similar to network 2, except that the fully connected layer 7 was removed. Network 3, with corrected layer numbering, is shown below.

```
# Layer 1
Convolution2D(96, 11, 11, input_shape = (28,28,1))
Activation('relu')
MaxPooling2D(pool_size=(2, 2))

# Layer 2
Convolution2D(256, 5, 5)
Activation('relu')
MaxPooling2D(pool_size=(2, 2))

# Layer 3
ZeroPadding2D((1,1))
Convolution2D(512, 3, 3)
Activation('relu')

# Layer 4
ZeroPadding2D((1,1))
Convolution2D(1024, 3, 3)
Activation('relu')

# Layer 5
ZeroPadding2D((1,1))
Convolution2D(1024, 3, 3)
Activation('relu')
MaxPooling2D(pool_size=(2, 2))

# Layer 6
Flatten()
Dense(1024)
Activation('relu')
Dropout(0.5)

# Layer 7
Dense(2)
```

3.3.2.4 Network 4

This network was similar to network 3, except that the convolutional layer 4 was removed. Network 4, with corrected layer numbering is shown below.

```
# Layer 1
Convolution2D(96, 11, 11, input_shape = (28,28,1))
Activation('relu')
MaxPooling2D(pool_size=(2, 2))

# Layer 2
Convolution2D(256, 5, 5)
Activation('relu')
MaxPooling2D(pool_size=(2, 2))

# Layer 3
ZeroPadding2D((1,1))
Convolution2D(512, 3, 3)
Activation('relu')

# Layer 4
ZeroPadding2D((1,1))
Convolution2D(1024, 3, 3)
Activation('relu')
MaxPooling2D(pool_size=(2, 2))

# Layer 5
Flatten()
Dense(1024)
Activation('relu')
Dropout(0.5)

# Layer 6
Dense(2)
```

3.3.2.5 Network 5

This network was similar to network 4, except that the number of convolution kernels in layer 4 was reduced from 1024 to 512. Network 5 is shown below.

```
# Layer 1
Convolution2D(96, 11, 11, input_shape = (28,28,1))
Activation('relu')
MaxPooling2D(pool_size=(2, 2))

# Layer 2
Convolution2D(256, 5, 5)
Activation('relu')
MaxPooling2D(pool_size=(2, 2))

# Layer 3
ZeroPadding2D((1,1))
Convolution2D(512, 3, 3)
Activation('relu')

# Layer 4
ZeroPadding2D((1,1))
Convolution2D(512, 3, 3)
Activation('relu')
MaxPooling2D(pool_size=(2, 2))

# Layer 5
Flatten()
Dense(1024)
Activation('relu')
Dropout(0.5)

# Layer 6
Dense(2)
```

3.3.3 Training and Testing

In these experiments, the synthetic *person 3* (recordings with the third depth level, see section 3.3.1) acted as the test set, while the rest of the persons acted as the training set. Cross-validation could have been used, as in the regional anesthesia experiments, to get more valid results, but this would have increased the training time by a factor of four. Since these were synthetic experiments, created to prepare the ground for the experiments on real ultrasound images, the benefit of reducing the training time was seen as more important.

In order to measure the average results over multiple runs, each experiment was run five times.

3.3.3.1 Loss Function

The MSE loss function, used in the 1D synthetic ultrasound experiments (Eq. 2-2), was used here as well. However, since the location of each image was now described in two dimensions, the image labels were now vectors rather than scalars, and the loss function was extended as described in section 2.1.3.1 (Eq. 2-3). Consequently, the loss function used in these experiments was

$$\frac{1}{2N}\sum_{i}\sum_{j=1}^{2}\left(y_{ij}-prediction_{ij}\right)^{2}$$

, where i is the current example, and j is the dimension in the label.

3.3.3.2 Test Function

The naïve choice for a test function would be to extend the MAE function used in the 1D synthetic ultrasound experiments to two dimensions, in the same way as the MSE function was extended. The resulting function, shown in section 2.1.3.1 (Eq. 2-5), is repeated here, with M replaced by 2.

$$\frac{1}{2N}\sum_{i}\sum_{j=1}^{2}\left|y_{ij}-prediction_{ij}\right|$$

However, the extended MAE is misleading when one considers the goal of the task – to guide the user as close as possible to the correct location. Because of the geometric nature of the task, the real distance to the correct location is the *Euclidean* distance, but MAE produces half of the *Manhattan* distance. For example, an error of $x, y = 2, 2$ gives half a Manhattan distance of 2, and a Euclidean distance of about 2.83. In contrast, an error of $x, y = 0, 4$ also gives half a Manhattan distance of 2, but a Euclidean distance of 4. Clearly, the first example is closer to the correct location, but the MAE fails to recognize this. The consequence of testing with the MAE is that when comparing the results of two models, the wrong conclusion can be taken as to which model is better. The MAE was therefore neither used as the loss function nor as the test function in these experiments.

The MSE, however, is directly related to the Euclidean distance. Since the MSE of a single example with errors e_x and e_y is $\frac{e_x^2+e_y^2}{2}$, the Euclidean distance d can be found by multiplying the MSE with 2 and taking the square root, giving $d = \sqrt{e_x^2 + e_y^2}$. Because of this, both the MSE and the Euclidean distance could be used as the loss function, but the MSE was chosen, to avoid the computationally costly square roots.

To calculate the average of the actual distances between the correct image locations and the predicted locations, a *Euclidean distance* function was developed:

$$\frac{1}{n}\sum_{i}\sqrt{\sum_{j=1}^{2}\left(y_{ij} - prediction_{ij}\right)^2} \tag{3-2}$$

This function is similar to the extended MSE, except that a root is taken on the inner summation, resulting in the Euclidean distance between the two locations. Also, the mean of the Euclidean distances is taken over the whole batch, by dividing by n instead of $2n$.

The Euclidean distance metric was applied in all the experiments. The last three experiments additionally measured the MAE in each dimension separately. The two functions used to measure the MAE are here called MAEX (lateral direction), and MAEY (longitudinal direction).

3.3.3.3 Scaling the Dataset Labels

The labels of the images – the ultrasound probe positions – were normalized to lie in the range $[0,1]$ in both dimensions. Training with these labels caused a problem, since it made it seem like the rectangle, which the probe moved inside, was squared. As a result, for a distance error e_x in the x-direction and a distance error e_y in the y-direction, if $e_x = e_y$ then the two errors would be treated as equally bad. However, the actual movement range of the probe might not be equal in both dimensions. If we assume that the probe in these synthetic experiments moved twice as long in the y-direction than in the x-direction, the situation would be like in Figure 3.11.

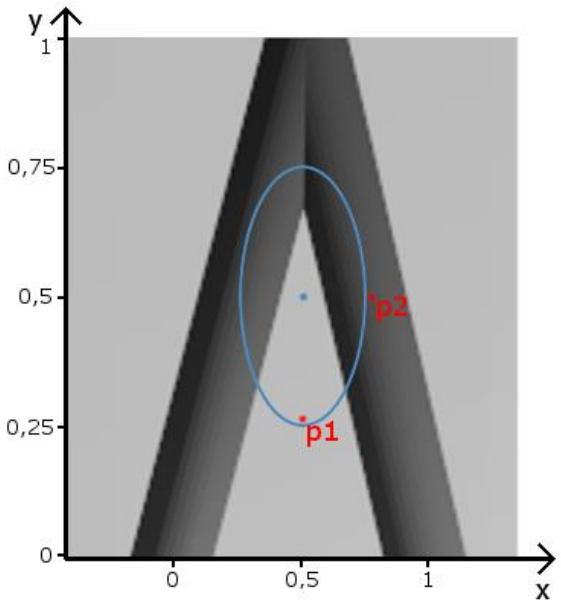

Figure 3.11: The effect of not compensating for the different scales on the two axes. Points at equal distance from the central blue dot form an ellipse instead of a circle.

In Figure 3.11, the blue dot at the center represents the correct label of an image. All positions that are at a distance 0.25 from the correct position are shown as a blue ellipse. The elliptic shape is a result of the different scales on the two axes. Suppose a neural network predicts the position labelled p_2. Then, in order to decrease the Euclidean distance loss function, the network will try to change its weights so that a position on the inside of the blue ellipse is predicted instead. Since p_1, for example, is on the inside of the ellipse, the network will prefer to predict this position, but in reality, p_1 is further away from the correct position than p_2.

One way to correct this would be to change the scale of the y-axis, so that its range was [0,2], and re-label the y-coordinate of all images accordingly. However, this solution was not chosen because in the general case, normalized values are more portable. If the labels were to be permanently stored together with the image data, and if the dataset were to be used in a different setting at a later point, non-normalized labels might not be appropriate for that setting.

Instead, the y-coordinate of the image labels was scaled during run-time using the loss- and/or test functions. The scale factors used in these functions are here called $yScale_{loss}$ and $yScale_{test}$. Equation 3-3 shows the MSE loss function modified with such scaling.

$$\frac{1}{2n}\sum_{i}\sum_{j=1}^{2}\left(yScale_{loss}\left(y_{ij}-prediction_{ij}\right)\right)^{2} \tag{3-3}$$

Setting $yScale_{loss} = 1$ would produce the situation shown in Figure 3.11 above, while setting $yScale_{loss} = 2$ would produce the situation shown in Figure 3.12, where the points at distance 0.25 from the correct position form a circle.

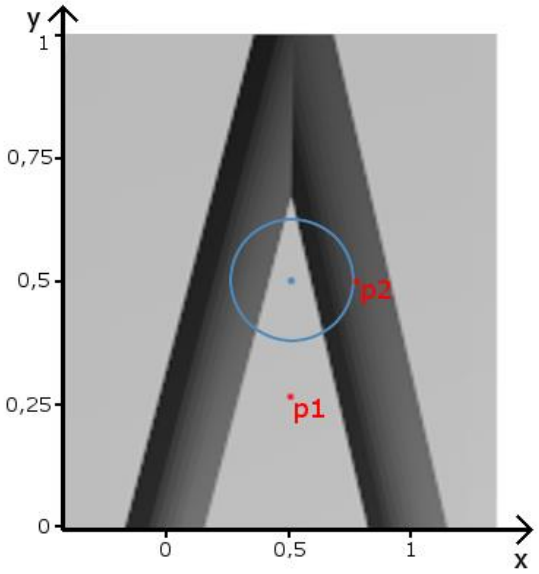

Figure 3.12: The effect of compensating for the different scales on the two axes. Points at equal distance from the central blue dot now form a circle.

The same concept was applied to the Euclidean distance test function:

$$\frac{1}{N}\sum_{i}\sqrt{\sum_{j=1}^{2}\left(yScale_{test}\left(y_{ij}-prediction_{ij}\right)\right)^{2}} \tag{3-4}$$

The concept of scaling the y-coordinate was applied in the last three experiments. First, in exp. 14, $yScale_{test}$ was set to 2, while $yScale_{loss}$ remained 1. This means that the Euclidean distance produced during testing was corrected for differences in scale, while the training was done without this correction. This was done to see the effect of changing $yScale_{loss}$ in the subsequent experiments. Second, in exp. 15, both $yScale_{test}$ and $yScale_{loss}$ were set to 2, which means that both the training and the testing function used the correct scaling. Last, in exp. 16, $yScale_{test}$ was still 2, but $yScale_{loss}$ was increased to 4. This was done to see the effect of over-focusing on the y-coordinate. Doing so might be useful in a situation where it is more important to obtain a correct y-coordinate, than a correct x-coordinate. In ultrasound imaging, this might sometimes be the case, since, when moving the probe sideways, most of the features remain visible and unchanged, although shifted sideways. On the other hand, when

moving the probe in the longitudinal direction, new slices of the body are displayed, and features may change fast. Accordingly, it might be beneficial to focus more attention on learning the correct y-coordinate, than on learning the x-coordinate. The scaling technique described above is an easy way to test different focuses.

3.3.4 Experiment Descriptions

3.3.4.1 Experiment 1 (simple network)

This experiment used the simple neural network described in section 3.3.2.1. It acted as a benchmark to measure the rest of the experiments against.

Property	**Value**
Network	Network 1 (section 3.3.2.1)
Total number of images	684
Training images	513
Test images	171
Test images source	Person 3
Runs	5
Epochs	20
Batch size	10
Loss function	Mean squared error (Eq. 2-3)
Test function	Euclidean distance (Eq. 3-2)
Optimizer	Adam optimizer
Learning rate	10^{-5}
Decay	0
L2-regularization	0

Table 3-5: Properties of 2D synthetic ultrasound experiment 1.

3.3.4.2 Experiment 2 (AlexNet)

The purpose of this experiment was to see the effect of moving from a simple, custom network to a larger, well-known network, like AlexNet. The experiment was similar to exp. 1, except that network 2 (section 3.3.2.2) was used.

3.3.4.3 Experiment 3 (remove layer 7)

This experiment aimed at reducing the training time by removing one layer, while maintaining the precision. The experiment was similar to exp. 2, except that network 3 (section 3.3.2.3) was used.

3.3.4.4 Experiment 4 (remove layer 4)

This experiment aimed at further reducing the training time by removing another layer, while maintaining the precision. The experiment was similar to exp. 3, except that network 4 (section 3.3.2.4) was used.

3.3.4.5 Experiment 5 (reduce conv. kernels)

The purpose of this experiment was again to reduce the training time, but this time by reducing the number of convolution kernels. The experiment was similar to exp. 4, except that network 5 (section 3.3.2.5) was used.

3.3.4.6 Experiment 6 (increased learning rate)

This experiment was set up to see if the network could benefit from a larger learning rate. The experiment was similar to exp. 5, except that the learning rate was increased to 10^{-4}.

3.3.4.7 Experiment 7 (highest learning rate)

This experiment was similar to exp. 6, except that the learning rate was further increased to 10^{-3}.

3.3.4.8 Experiment 8 (apply decay)

The purpose of this experiment was to see the effect of introducing learning rate decay. The experiment was similar to exp. 6 (not exp. 7), except that a decay of $4\ x\ 10^{-6}$ was applied.

With this amount of decay, the learning rate would be reduced to 20% of its original value by the end of the 20th epoch.

3.3.4.9 Experiment 9 (increased decay)

This experiment was similar to exp. 8, except that the decay was increased to $4.5\ x\ 10^{-6}$.

With this amount of decay, the learning rate would be reduced to 10% of its original value by the end of the 20th epoch.

3.3.4.10 Experiment 10 (smaller batch size)

The purpose of this experiment was to see the effect of changing the batch size, to possibly find a better balance between precision and training time. In this experiment, the changes made to the decay were rolled back so that the settings were similar to exp. 6, except that the batch size was reduced to 5.

3.3.4.11 Experiment 11 (larger batch size)

This experiment was similar to exp. 10, except that the batch size was increased to 20.

3.3.4.12 Experiment 12 (apply L^2)

In this experiment, L^2-regularization was applied to the network. The changes to the batch size were rolled back, and this experiment was again based on exp. 6, except that a L^2-regularization of 0.001 was applied.

The L^2-regularization was applied to all four convolution layers, as well as to the fully connected sub-layer in layer 5.

3.3.4.13 Experiment 13 (increased L^2)

This experiment was similar to exp. 12, except that the L^2-regularization was increased to 0.01.

3.3.4.14 Experiment 14 ($yScale_{loss}$ 1)

This experiment was again based on exp. 6, by rolling back the changes made to the L2-regularization in the previous experiments. Next, the factors $yScale_{loss}$ and $yScale_{test}$ were applied to the MSE loss function and to the Euclidean distance test function, as described in section 3.3.3.3. Also, two additional metrics were used during testing: MAEX and MAEY. Finally, the training was done over 40 epochs instead of 20, in case the effect of scaling needed more time to become visible. The changes, compared to exp. 6, are shown below.

Property	**Value**
Loss function	Mse with y-coordinate scaling (Eq. 3-3)
Test function	Euclidean distance with y-coordinate scaling (Eq. 3-4) MAEX (Eq. 2-4) MAEY (Eq. 2-4)
$yScale_{loss}$	1
$yScale_{test}$	2
Epochs	40

Table 3-6: Differences between 2D synthetic ultrasound experiments 6 and 14.

3.3.4.15 Experiment 15 ($yScale_{loss}$ 2)

This experiment was similar to exp. 14, except for an increased scaling of the y-coordinate during training. The two scaling factors are shown below.

Property	**Value**
$yScale_{loss}$	2
$yScale_{test}$	2

Table 3-7: Scaling factors used in 2D synthetic ultrasound experiment 15.

3.3.4.16 Experiment 16 ($yScale_{loss}$ 4)

This experiment was similar to exp. 15, except for a further increased scaling of the y-coordinate during training. The two scaling factors are shown below.

Property	Value
$yScale_{loss}$	4
$yScale_{test}$	2

Table 3-8: Scaling factors used in 2D synthetic ultrasound experiment 16.

3.4 Lower Arm Dataset: Regression in Two Dimensions

In these experiments, a neural network was trained to predict the two-dimensional acquisition location of real ultrasound images. The goal was to see how accurate the location would be predicted when using the same methods as in the 1D and 2D synthetic experiments.

As the size of the dataset was far greater than in the previous experiments, training time became an issue, and testing many small adjustments of hyperparameters, was infeasible. Instead, the network topology and the hyperparameters found to be optimal in the 2D synthetic experiments in section 3.3 were used as a starting point. There is, however, no guarantee that these hyperparameters were optimal in this new context, and the results may have been improved if thorough hyperparameter testing had been conducted.

The results of the initial setup were compared to an experiment with a pre-trained network. Finally, the effect of changing the y-coordinate scaling factor was tested, like in the 2D synthetic experiments in section 3.3.

3.4.1 Description of Dataset

The dataset used in these experiments consisted of real ultrasound images acquired from the right lower arm of nine test persons. During the recordings, the lower arm of the test person was rested on a flat, horizontal surface, with the palm facing upwards. Recordings were performed in a straight line, starting at the inside of the elbow and ending at the wrist joint. The

first image in each recording sequence was given the y-coordinate 0, and the last image was given the y-coordinate 1. The y-coordinates for the rest of the images were interpolated between 0 and 1. For these labels to be correct, the movement speed of the ultrasound probe would have to be constant throughout the recording, for example by controlling the probe with a robotic arm. However, for these simple experiments, the probe movement was controlled by hand, and the operator tried to keep the speed constant.

The recordings were performed with five different lateral offsets, and these offsets were encoded as the x-coordinate in the images' labels. The first recording started on the left side of the elbow, when looking at the arm from the test person's perspective, and ended on the left side of the wrist joint. Images from this recording were given the x-coordinate 0. The next recording was offset by 45° about the main axis of the lower arm, in the clockwise direction, and images from this recording was given the x-coordinate 0.25. Each subsequent recording was offset by an additional 45°, so that, finally, the fifth recording started on the right side of the elbow, and ended on the right side of the wrist joint. Images from the last recording were thus given the x-coordinate 1. In Figure 3.13, the coordinate system is superimposed on an image of the lower arm, and the three first recording offsets are shown.

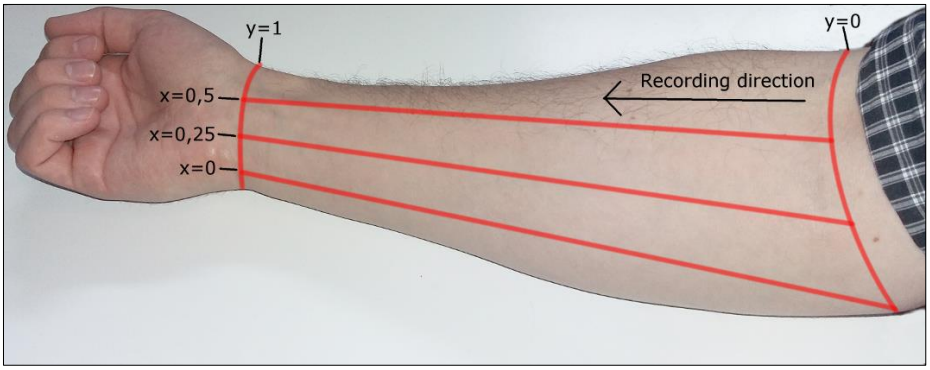

Figure 3.13: The coordinate system of the lower arm dataset.

In total, three similar recordings were taken for each offset, five offsets were used per person, and nine persons were scanned, resulting in 135 recordings. Each recording consisted of between 120 and 250 images, but to ensure that all offsets were equally trained on, 100 images from each recording were extracted, evenly spread out over the length of the recording. In this extraction, the first and the last image of the original recording were always preserved, so that they could correctly be given the y-coordinates 0 and 1, respectively. The final dataset consisted of 13 500 images.

The average length of the lower arm of the test persons, measured from the inside of the elbow to the wrist, was $24\ cm$. The average distance between the first and the last offset (half of the circumference of the lower arm) was estimated to be about $12.5\ cm$. However, note that the lower arm is considerably thicker near the elbow than at the wrist, so $12.5\ cm$ would be most correct at the middle of the lower arm's length, where $y = 0.5$. An approximation used in these experiments was that the distance between $y = 0$ and $y = 1$ was twice the distance between $x = 0$ and $x = 1$.

The size of the original images was 1193×798 pixels. All images were scaled to 100×64 pixels, to fit the neural network in section 3.3, and they were also scaled to 224×224 pixels, to fit the pre-trained neural network from section 3.1.

Examples of images from the beginning, center, and end of a recording from the middle offset are shown in Figure 3.14.

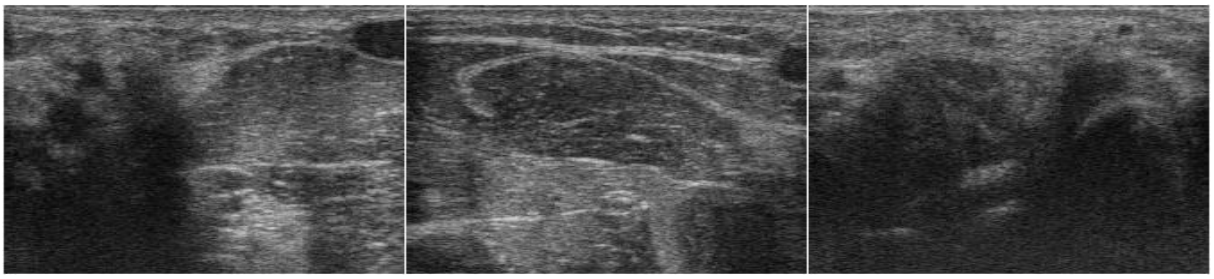

Figure 3.14: Examples of images from a recording in the lower arm dataset. (**Left**) First frame of the recording, near the elbow. (**Center**) Middle frame of the recording. (**Right**) Last frame of the recording, at the wrist.

3.4.2 Description of Networks

In these experiments, three different networks were used, all of them very similar to networks presented earlier. These will be described below:

3.4.2.1 Network 1

This network was used for training on and prediction of the y-coordinate only. The network was identical to network 5 of the 2D synthetic experiments in section 3.3, i.e. the network found after testing different variations to the network topology. The only difference was the change of the number of output nodes from two to one, as shown below:

```
# Layer 6
Dense(1)
```

3.4.2.2 Network 2

Network 2 was almost identical to the network used in the regional anesthesia experiments in section 3.1 – a network based on the VGG16 network, described in appendix 9.3. The purpose of using this network was to apply pre-training. As in section 3.1, layers 1 through 5 were kept intact, in order to copy weights from the pre-trained VGG16 network, but the final layers were changed, as shown below:

```
# Layer 6
Flatten()
Dense(512)
Activation('relu')
Dropout(0.5)

# Layer 7
Dense(512)
Activation('relu')
Dropout(0.5)

# Layer 8
Dense(1)
```

The changes were to reduce the number of nodes in layers 6 and 7, to reduce the training time, and to only have one node in the output layer, since this network was intended to only learn one coordinate dimension.

3.4.2.3 Network 3

Network 3 was identical to network 2, except that the output layer contained two nodes instead of one, in order to predict two coordinate dimensions.

3.4.3 Training and Testing

The separation of the dataset into a training set and a test set was done in two different ways in these experiments. In some of the experiments, every 10^{th} image from every recording was used as the test set, while the rest of the images were trained upon. In the other experiments, the test set consisted of all images from one person, while the rest of the persons were trained upon. This arrangement was repeated according to the principle of cross-validation, so that every person in turn acted as the test set, and the average over all runs could be computed.

Another difference between the experiments was the choice of image labels. In the first experiments, only recordings from the center offset were used, and the image labels only consisted of the y-coordinate. In this aspect, these experiments were similar to the 1D synthetic experiments in section 3.2, which focused on regression training in one dimension. In the last experiments, recordings from all offsets were used, and the image labels consisted of both the y- and the x- coordinate, which made these experiments similar to the 2D synthetic experiments in section 3.3.

3.4.3.1 Loss and Test Function

The loss and test functions were identical to those presented in the synthetic experiments. On the experiments with a single coordinate dimension, MSE (Eq. 2-2) was used for training, and MAE (Eq. 2-4) was used for testing. On the experiments with two coordinate dimensions, extended MSE with y-coordinate scaling (Eq. 3-3) was used for training, and Euclidean distance with y-coordinate scaling (Eq. 3-4) was used for testing.

In all the experiments with two coordinate dimensions, $yScale_{test} = 2$ was used. In the same experiments, $yScale_{loss}$ was varied between one, two and four, to observe the effect.

3.4.4 Experiment Descriptions

3.4.4.1 Experiment 1 (1D: no pre-training)

Experiment 1 used Network 1 along with the hyperparameters found to be optimal during the 2D synthetic experiments. This network was trained on the center offset recordings only, using every 10^{th} image as the test set. The purpose of using this network was to see how it compared to using a pre-trained network (exp. 2). The parameters used are shown below.

Property	**Value**
Network	Network 1 (section 3.4.2.1)
Total number of images	2700 (only center offset)
Training images	2430
Test images	270
Test images source	Every 10^{th} image from each person
Coordinate dimensions	1 (only y-coordinate)
Runs	5
Epochs	20
Batch size	10
Loss function	MSE (Eq. 2-2)
Test function	MAE (Eq. 2-4)
Optimizer	Adam optimizer
Learning rate	10^{-4}
Decay	0
L2-regularization	0

Table 3-9: Properties of lower arm experiment 1.

3.4.4.2 Experiment 2 (1D: pre-training)

This experiment was similar to exp. 1, except that a pre-trained network was used.

Property	Value
Network	Network 2 (section 3.4.2.2)
Layers frozen from training	1-5

Table 3-10: Differences between lower arm experiments 1 and 2.

3.4.4.3 Experiment 3 (1D: layer 5 unfrozen)

This experiment was similar to exp. 2, except that layer 5 (3 convolutional sub-layers and a max-pooling sub-layer) were unfrozen so that they could be adjusted during training.

3.4.4.4 Experiment 4 (1D: cross-validation)

In this experiment, cross-validation was used instead of testing on every 10^{th} image. The experiment was similar to exp. 3, except as shown below:

Property	Value
Training images	2400
Test images	300
Test images source	One person, cross-validation

Table 3-11: Differences between lower arm experiments 3 and 4.

3.4.4.5 Experiment 5 (2D: train and test on all persons)

From this experiment, and throughout the rest of the lower arm experiments, the network was trained on both coordinate dimensions. Exp. 5 again used every 10th image as the test set. Table 3-12 shows the property changes from exp. 4.

Property	Value
Network	Network 3 (section 3.4.2.3)
Total number of images	13 500 (all offsets)
Training images	12 150
Test images	1350
Test images source	Every 10th image from each person
Coordinate dimensions	2 (x- and y-coordinate)
Loss function	Extended MSE with y-coordinate scaling (Eq. 3-3)
Test function	Euclidean distance with y-coordinate scaling (Eq. 3-4)
$yScale_{loss}$	2
$yScale_{test}$	2

Table 3-12: Differences between lower arm experiments 4 and 5.

3.4.4.6 Experiment 6 (2D: $yScale_{loss}$ 1)

Experiment 6 was similar to exp. 5, except that the test set consisted of all images from one person, and each person took turns acting as the test set (cross-validation). Also, $yScale_{loss}$ was set to 1.

Property	Value
Training images	12 000
Test images	1500
Test images source	One person, cross-validation
$yScale_{loss}$	1

Table 3-13: Differences between lower arm experiments 5 and 6.

3.4.4.7 Experiment 7 (2D: $yScale_{loss}$ 2)

The only change from exp. 6 to exp. 7 was to increase $yScale_{loss}$ to 2.

3.4.4.8 Experiment 8 (2D: $yScale_{loss}$ 4)

In exp. 8, $yScale_{loss}$ was increased further to 4.

3.5 Lower Spine Dataset: Classification of Vertebrae

Palpation is commonly used to identify a specific vertebra, by starting from the sacrum and counting the vertebrae along the spine. An alternative way to do this could be to place an ultrasound probe on the sacrum, moving it upwards along the spine, and counting the vertebrae as they appear on the screen. Such counting could be supported by a neural network capable of determining whether the current ultrasound image depicted a vertebra or not. Testing such a neural network was the purpose of the experiment in this section. A single experiment was performed, where the network was trained to classify images from the lower spine as sacrum, vertebra, or intervertebral disc images (the gap between the vertebrae).

3.5.1 Description of Dataset

The dataset used in this experiment consisted of ultrasound images taken from the lower back of three test persons. The probe was placed over the sacrum, at the center of the spine, and oriented in the transverse plane. When the recording started, the probe was moved upwards along the spine, while counting the vertebrae of the lower spine (the Lumbar vertebrae). After five lumbar vertebrae had been counted, the recording was stopped.

The images were manually divided into three categories: Sacrum, vertebra, and gap. In Figure 3.15, examples of images from the categories are shown, with red marking superimposed. To the left is a typical image of the sacrum, recognizable by the horizontal white border in the center of the image that signals a flat bone structure beneath. In the center, a vertebra image is shown, with red marking around the top border of the spinous process bone structure that extends from the vertebra towards the surface. The dark shadow of the bone can be seen underneath. The image to the right is taken between two vertebrae, and has no clear bone

structure borders. The images were scaled to 224×224 to match the neural network's input layer.

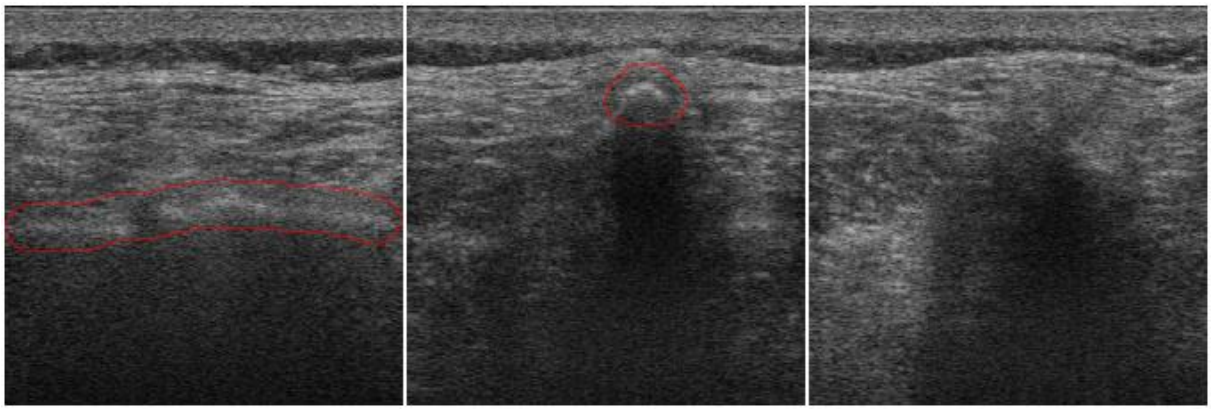

Figure 3.15: Examples of ultrasound images in the lower spine dataset.(**Left**) Image of the sacrum, with markings around the flat bone structure. (**Center**) Image of a vertebra, with markings around the top of the spinous process bone structure. (**Right**) Image taken between two vertebrae.

Three similar recordings were performed for each person, resulting in nine recordings in total. The length of the recordings varied from 348 to 483 images, and the total number of images from all recordings was 3619. Some of these images were not used for training or testing, however, because it was hard to determine the correct category. Only the recording intervals that could be categorized with confidence was included, and even then, the boundaries between the regions were hard to determine exactly. Specifically, the transition from the sacrum to the first gap or vertebra was hard to identify, and therefore only four (and in some cases three) vertebra regions have been identified in the recordings.

The number of images in each class was unbalanced, as the regions containing the vertebrae was larger than the gap regions and the sacrum region. In total from all recordings, 410 images of the sacrum, 1775 images of vertebrae, and 677 images of gaps were identified and included in the training and testing, which sums to 2862 images altogether.

3.5.2 Description of Network

The network used in this experiment was based on the VGG16 network (appendix 9.3). Layers 1-5 were kept intact and loaded with pre-trained weights, while the final layers were replaced with the layers shown below:

```
# Layer 6
Flatten()
Dense(512)
Activation('relu')
Dropout(0.5)

# Layer 7
Dense(512)
Activation('relu')
Dropout(0.5)

# Layer 8
Dense(3)
Activation('softmax')
```

3.5.3 Training and Testing

In this experiment, images from two persons were used for training, and the images from the last person were used for testing. This was applied with cross-validation, so that the average performance could be found.

As in the regional anesthesia experiments, the cross-entropy (Eq. 2-1) was used as the loss function, and the accuracy measure (Eq. 3-1) was used as the test function. Five runs were performed for each choice of test person in the cross-validation scheme.

3.5.4 Experiment Description

Property	Value
Network	VGG-like (section 3.5.2)
Total number of images	2862
Training images	Variable. Images from two persons
Test images	Variable. Images from one person
Runs	5
Epochs	20
Batch size	10
Loss function	Cross-entropy (Eq. 2-1)
Test function	Accuracy (Eq. 3-1)
Optimizer	Adam optimizer
Learning rate	10^{-4}

Table 3-14: Properties of the lower spine experiment.

3.6 Technical Set Up

Tensorflow (Google, 2016) was chosen as the neural network framework in this thesis, because of its popularity in the field. The version `Tensorflow 0.11.0rc0` was used – a version capable of utilizing the GPU for computations.

On top of Tensorflow, the high-level neural network library Keras was used (Keras, 2017). Keras is considerably easier to work with than Tensorflow, because it requires a lot less code to perform the same operations as in Tensorflow, while the full functionality of Tensorflow is still available if required. Also, Keras was recently adopted by Google as being an official part of Tensorflow.

Linux (Ubuntu 14.04) was chosen as the operating system, because it works most seamlessly together with Tensorflow.

While the underlying computations of Tensorflow are written in c++, Python is the development language used to control and run both Tensorflow and Keras. Versions 3.4 and 3.5 were used in this setup.

To utilize the GPU, Tensorflow relies on two additional components, CUDA and cuDNN. CUDA is NVIDIA's parallel computing platform, enabling programs to be run on the GPU. In this set up, CUDA 7.5 was used. cuDNN is also made by NVIDIA, and is a library of primitives specifically designed for deep neural networks in the CUDA context. cuDNN 5.1 was used in this set up.

Scaling and cropping of images at runtime was done using the Python library Pillow (Lundh & Clark, 2016).

Some of the ultrasound images acquired were in a raw format. To visualize them, the Python library PyPNG (Jones, 2016) was used to convert them to png-files. This made it easier to verify the results of the training and testing.

4 Results

In this chapter, the results of all experiments are presented. The chapter is divided into separate sections for each dataset.

4.1 Regional Anesthesia Dataset

4.1.1 Experiment 1 (cross-validation, no pre-training)

The average accuracy over five runs at the end of the 10th epoch is shown in Table 4-1, for each choice of test person, and for the total average over all test persons.

Person 1	Person 2	Person 3	Person 4	Average
0.756	0.901	0.609	0.680	0.737

Table 4-1: Accuracies of regional anesthesia experiment 1. The accuracies were averaged over five runs at the end of the 10th epoch.

This means that, on average, the network was able to correctly classify about 74% of the unseen images after the 10th epoch. The development of the accuracy over the epochs is shown in Figure 4.1, along with the average of the four test persons. The accuracies of each choice of test person increased slightly over the epochs, with the exception of person 2.

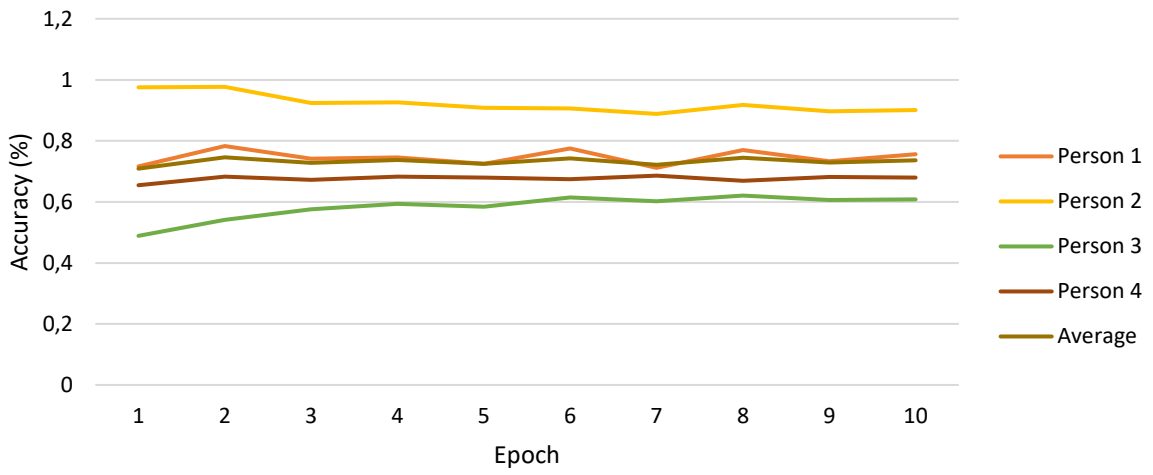

Figure 4.1: Classification accuracy in regional anesthesia experiment 1. The network was trained from scratch.

Due to the high accuracy of person 2 in the beginning, the best average accuracy over all test persons was achieved after the second epoch. The average accuracy was then 0.746, slightly higher than the accuracy after the 10th epoch.

4.1.2 Experiment 2 (pre-training applied)

The average accuracy over five runs at the end of the 10th epoch is shown in Table 4-2, for each choice of test person.

Person 1	Person 2	Person 3	Person 4	Average
0.922	0.917	0.826	0.746	0.853

Table 4-2: Accuracies of regional anesthesia experiment 2. The accuracies were averaged over five runs at the end of the 10th epoch.

This means that, on average, the network was able to correctly classify about 85% of the unseen images – a large increase compared to exp. 1 (no pre-training). The development of the accuracy over the epochs is shown in Figure 4.2, along with the average of the four test persons. The accuracies are stable or increase slightly over the epochs, with the notable exception of person 2 which has a significant accuracy reduction.

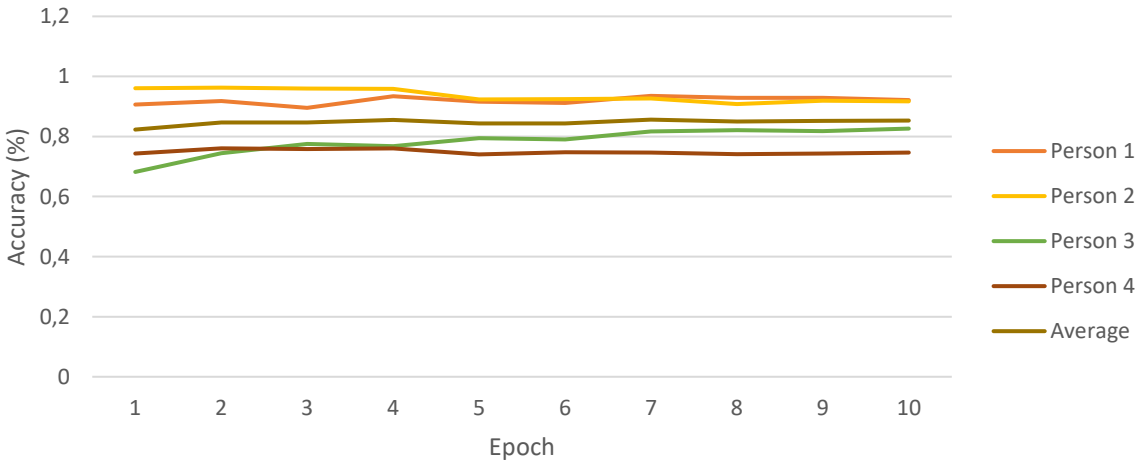

Figure 4.2: Classification accuracy in regional anesthesia experiment 2. The first five layers were loaded with pre-trained weights.

Figure 4.3 shows a comparison between the average accuracies in exp. 1 and exp. 2 (without and with pre-training). Pre-training seems to have a positive effect on the accuracy.

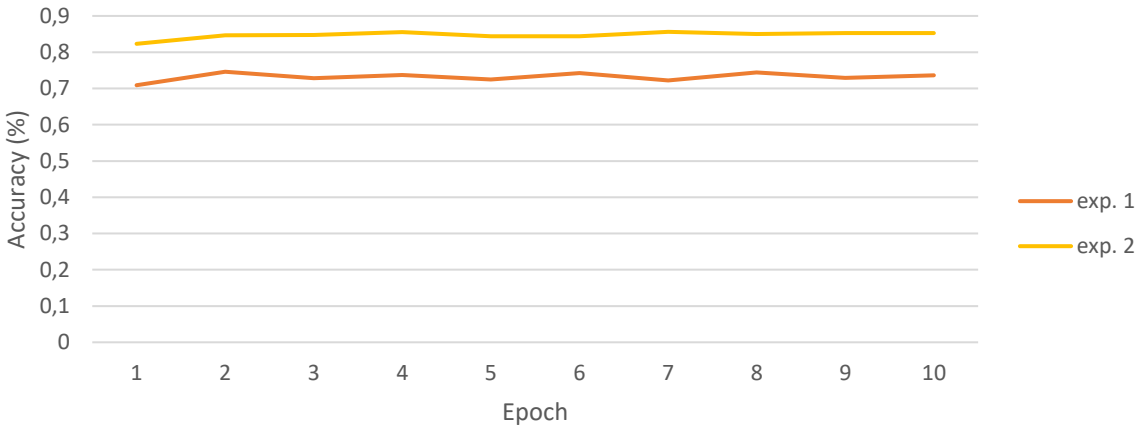

Figure 4.3: Comparison of average accuracies in regional anesthesia experiments 1 and 2.Pre-training was applied to exp. 2.

4.1.3 Experiment 3 (train and test on all persons)

The average classification accuracy after the 10th epoch was 0.996. This means that, when allowed to train on images very similar to those in the test set, the network correctly classified 239 of 240 images. The development of the accuracy over the epochs is shown in Figure 4.4.

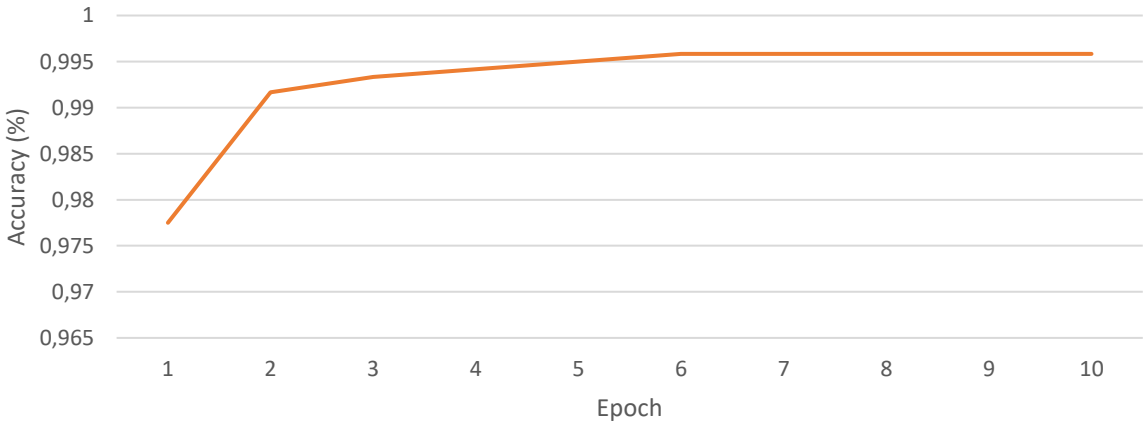

Figure 4.4: Classification accuracy in regional anesthesia experiment 3. The network was trained on nine of ten images from all persons.

4.1.4 Experiment 4 (remove person 3)

The average accuracy over five runs at the end of the 10th epoch is shown in Table 4-3, for each choice of test person, and for the total average over all test persons. By removing person 3, a significant improvement is achieved for all remaining test persons (compare with Table 4-1).

Person 1	Person 2	Person 4	Average
0.791	0.946	0.714	0.817

Table 4-3: Accuracies of regional anesthesia experiment 4. The accuracies were averaged over five runs at the end of the 10th epoch.

The development of the accuracy over the epochs is shown in Figure 4.5, along with the average of the three test persons. The average accuracy is still almost stable over the epochs, but a significant positive shift has been gained, compared to exp. 1. This is visible in Figure 4.6, which shows a comparison between the average accuracies in exp. 1 and exp. 4 (with and without person 3):

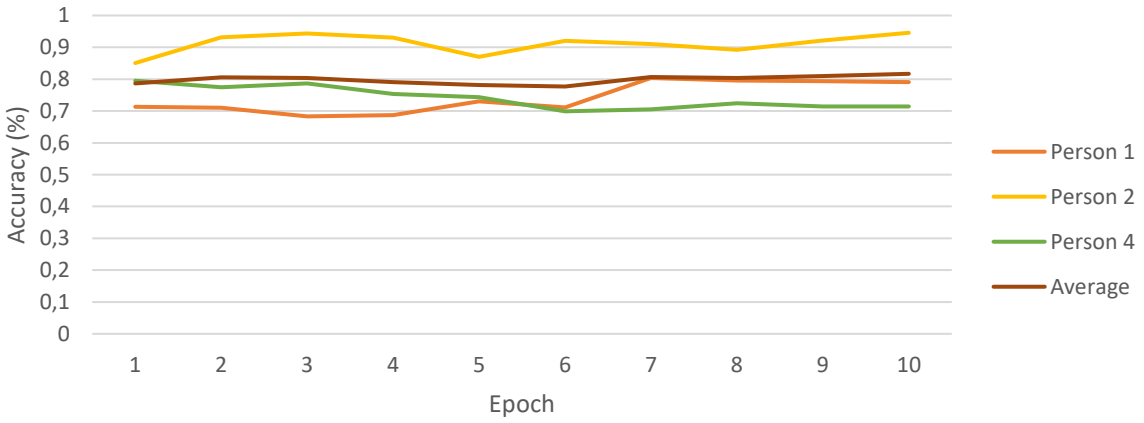

Figure 4.5: Classification accuracy in regional anesthesia experiment 4. Person 3 was excluded in this experiment.

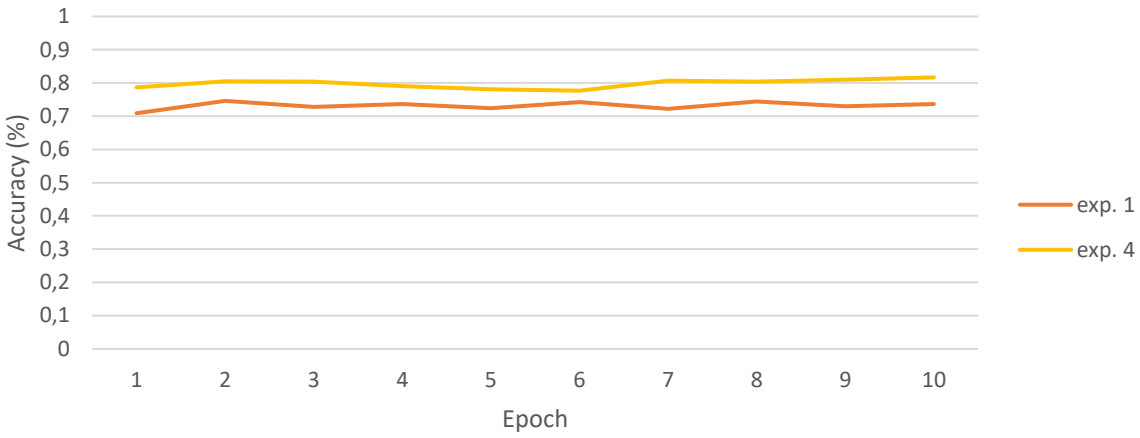

Figure 4.6: Comparison of average accuracies in regional anesthesia experiments 1 and 4. Person 3 was excluded in exp. 4.

4.2 1D Synthetic Ultrasound Dataset

4.2.1 Experiment 1 (no data-augmentation)

The MAE at the end of the 100th epoch is shown in Table 4-4.

Run1	Run2	Run3	Run4	Run5	Average
0.0607	0.0616	0.0286	0.0955	0.0338	0.0561

Table 4-4: MAE at the end of the 100th epoch in 1D synthetic ultrasound experiment 1.

This means that, on average, the trained network's predicted image locations were wrong by about 5.6% of the total recording distance. The development of the MAE over the epochs is shown in Figure 4.7, where the red line shows the average of the five runs. Most of the learning takes place during the first 30 epochs.

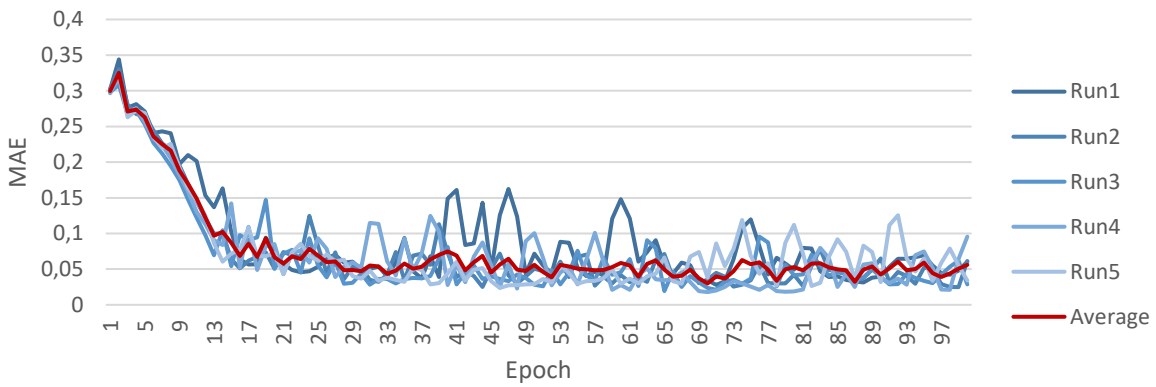

Figure 4.7: Average error in image location prediction - 1D synthetic ultrasound experiment 1.

The lowest MAE obtained during training was 0.0185 (run 4, epoch 79).

4.2.2 Experiment 2 (data augmentation)

The MAE at the end of the 100th epoch is shown in Table 4-5

Run1	**Run2**	**Run3**	**Run4**	**Run5**	**Average**
0.0470	0.0407	0.0340	0.0478	0.0230	0.0385

Table 4-5: MAE at the end of the 100th epoch in 1D synthetic ultrasound experiment 2.

This means that, on average, the trained network's predicted image locations were wrong by about 3.9% of the total recording distance. The development of the MAE over the epochs is shown in Figure 4.8, where the red line shows the average of the five runs. Most of the learning takes place early, but the network spends somewhat more time to obtain the same accuracy as in exp. 1.

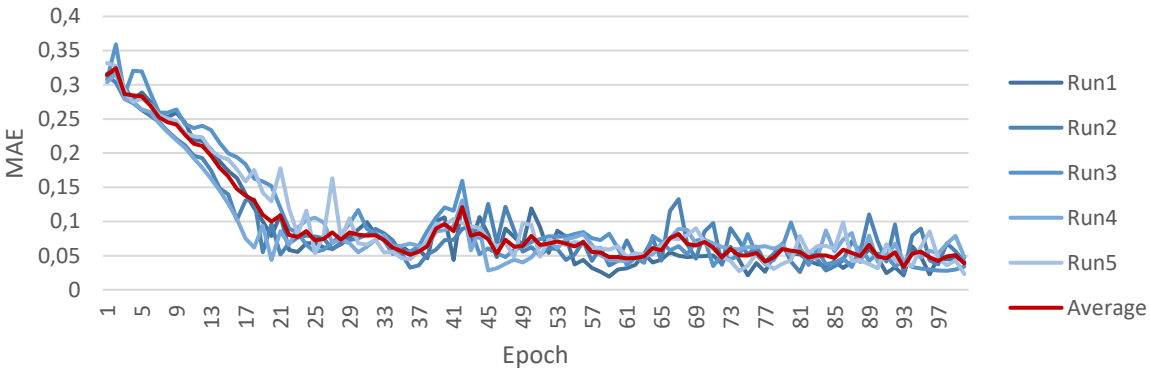

Figure 4.8: Average error in image location prediction - 1D synthetic ultrasound experiment 2. Data augmentation was applied in this experiment.

The lowest MAE obtained during training was 0.0193 (run 1, epoch 59).

Figure 4.9 shows a comparison between the average MAEs from exp. 1 and exp. 2 (without and with data augmentation). The network in exp. 1 learns faster in the beginning than the network in exp. 2. This is probably because, in contrast to the test images, the images produced by the data augmentation are not placed perfectly along the perfectly straight line.

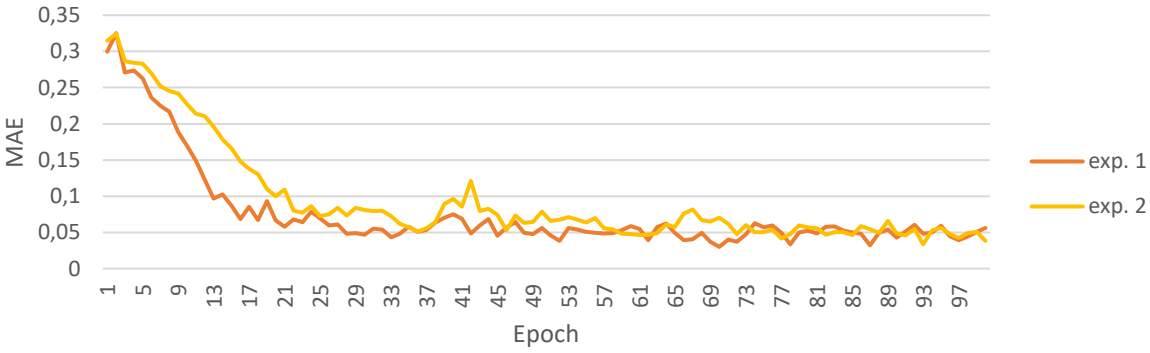

Figure 4.9: Comparison of average MAEs in 1D synthetic ultrasound experiments 1 and 2. Data augmentation was applied in exp. 2.

4.3 2D Synthetic Ultrasound Dataset

4.3.1 Experiments 1 and 2

The results of exp. 1 (simple network) and exp. 2 (AlexNet) are compared in Table 4-6. The first row of this table shows the average error (Euclidean distance) over the five runs at the end of the 20th epoch. The second row shows the smallest error obtained in any individual run, and at any epoch. The last row shows the average training time spent per epoch. A great reduction of error is achieved by using the larger network.

Measurement	**Exp. 1** Simple network	**Exp. 2** AlexNet-like
Average error over five runs after 20th epoch	0.213	0.0767
Smallest error obtained	0.202	0.0639
Training time per epoch	0.6 *s*	9.6 *s*

Table 4-6: Errors and training time in 2D synthetic ultrasound experiments 1 and 2.

The development of the prediction errors over the epochs is shown in Figure 4.10. The AlexNet clearly outperforms the simple network.

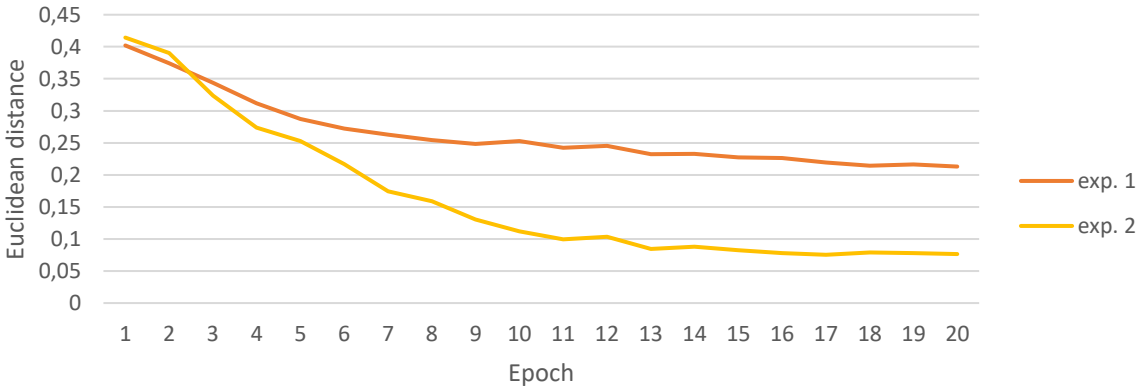

Figure 4.10: Average error in image location prediction - 2D synthetic ultrasound experiments 1 and 2. Exp. 1 used a simple, custom network with two convolutional layers, while the network in exp. 2 was based on the larger AlexNet, containing five convolutional layers.

4.3.2 Experiments 3, 4 and 5

Experiment 3 (remove layer 7), exp. 4 (remove layer 4) and exp. 5 (reduce conv. kernels) tested the effect on accuracy versus time as the neural network was simplified. The results of these experiments are shown in Table 4-7. The results from exp. 2 are also included, for comparison. Further experiments were based on exp. 5, since it achieved about the same accuracy as the AlexNet-like network in exp. 2, but with less than half the training time.

Measurement	**Exp. 2** AlexNet-like	**Exp. 3** Dropped a fc-layer	**Exp. 4** Dropped a conv-layer	**Exp. 5** Less conv-kernels
Average error over five runs after 20^{th} epoch	0.0767	0.0640	0.0714	0.0725
Smallest error obtained	0.0639	0.0529	0.0634	0.0654
Training time per epoch	9.6 *s*	9.0 *s*	6.3 *s*	3.7 *s*

Table 4-7: Errors and training time in 2D synthetic ultrasound experiments 2, 3, 4 and 5.

The development of the prediction errors over the epochs is shown in Figure 4.11. The networks learn at a different rate in the beginning, but achieve rather similar accuracies towards the end. The defining differences between them is therefore the training time, as seen in the previous table.

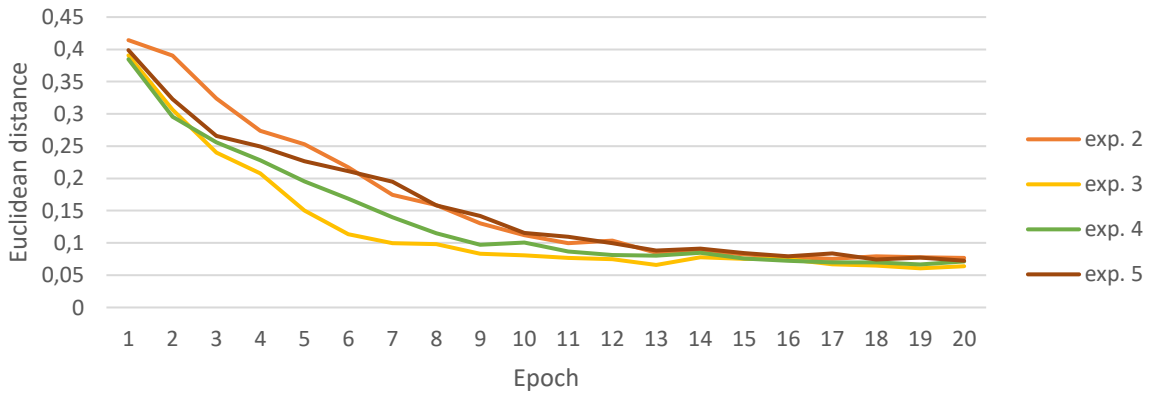

Figure 4.11: Average error in image location prediction - 2D synthetic ultrasound experiments 2-5. Compared to exp. 2, a fully connected layer was dropped in exp. 3, an additional convolutional layer was dropped in exp. 4, and the number of convolution kernels was reduced in exp. 5.

4.3.3 Experiments 6 and 7

Experiment 6 (increased learning rate) and exp. 7 (highest learning rate) tested the effect of increasing the learning rate (lr). The results of these experiments are shown in Table 4-8, along with the results from exp. 5 for comparison. The learning rate used in exp. 6 was superior to the value used in the other two experiments.

Measurement	**Exp. 5** lr $\mathbf{10^{-5}}$	**Exp. 6** lr $\mathbf{10^{-4}}$	**Exp. 7** lr $\mathbf{10^{-3}}$
Average error over five runs after 20th epoch	0.0725	0.0481	0.140
Smallest error obtained	0.0654	0.0353	0.0460

Table 4-8: Errors in 2D synthetic ultrasound experiments 5, 6 and 7.

The development of the prediction errors over the epochs is shown in Figure 4.12. The learning rate used in exp. 6 produced a steep learning curve in the beginning and also outperformed the other two experiments at the end of the 20th epoch. Raising the learning rate even further (exp. 7) had a negative effect on the accuracy.

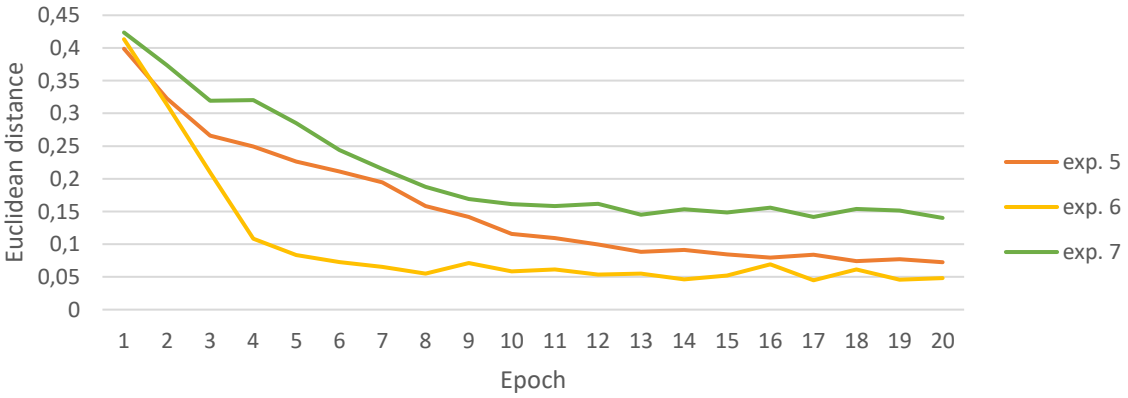

Figure 4.12: Average error in image location prediction - 2D synthetic ultrasound experiments 5-7. The learning rate in exp. 5, exp. 6, and exp. 7 was 10^{-5}, 10^{-4}, and 10^{-3}, respectively.

In exp. 7 (with the highest learning rate), the performances of the individual runs were diverse, as seen in Figure 4.13. This suggests that raising the learning rate too much can cause the learning process to overshoot the optimal weight values, as explained in section 2.1.5.3.

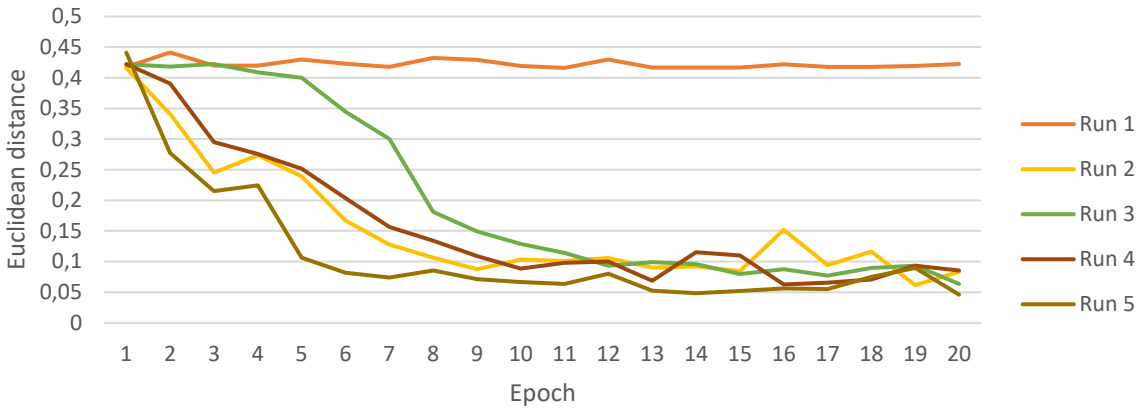

Figure 4.13: Average error in image location prediction - 2D synthetic ultrasound experiment 7.

4.3.4 Experiments 8 and 9

Experiment 8 (apply decay) and exp. 9 (increased decay) tested the effect of introducing decay on the learning rate. This was done since the learning rate of exp. 6 (the best so far) was very steep in the beginning, but became unstable towards the end. By reducing the learning rate towards the end, the goal was to reduce this instability. The results of exp. 8 and 9 are shown in Table 4-9, along with the results from exp. 6 for comparison.

Measurement	**Exp. 6** decay 0	**Exp. 8** decay $\boldsymbol{4\ x\ 10^{-6}}$	**Exp. 9** decay $\boldsymbol{4.5\ x\ 10^{-6}}$
Average error over five runs after 20th epoch	0.0481	0.0506	0.0461
Smallest error obtained	0.0353	0.0359	0.0358

Table 4-9: Errors in 2D synthetic ultrasound experiments 6, 8 and 9.

The development of the prediction errors over the epochs is shown in Figure 4.14. This figure shows how the decay produced smooth learning curves towards the end of the training; however, the decay did not increase the accuracies.

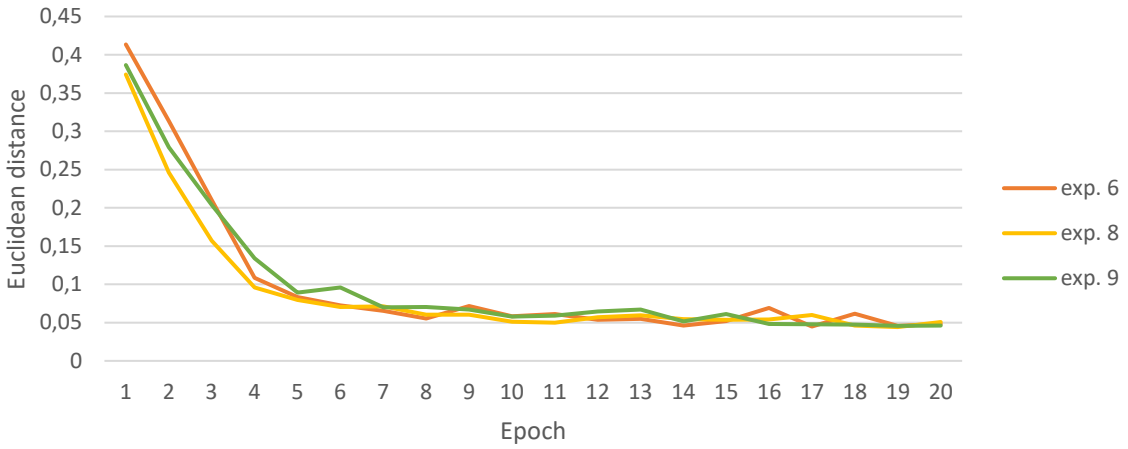

Figure 4.14: Average error in image location prediction - 2D synthetic ultrasound experiments 6, 8 and 9. The decay in exp. 6, exp. 8, and exp. 9 was zero, $4\ x\ 10^{-6}$, and $4.5\ x\ 10^{-6}$, respectively.

4.3.5 Experiments 10 and 11

Experiment 10 (smaller batch size) and exp. 11 (larger batch size) tested the effect of decreasing and increasing the batch size, to find a good balance between precision and training time. These experiments were again based on exp. 6, as the decay introduced in exp. 8 and 9 was reset to zero. The results of these experiments, including the training time per epoch, are shown in Table 4-10, along with the results from exp. 6 for comparison.

Measurement	**Exp. 6** batch size 10	**Exp. 10** batch size 5	**Exp. 11** batch size 20
Average error over five runs after 20^{th} epoch	0.0481	0.0462	0.0537
Smallest error obtained	0.0353	0.0308	0.0427
Training time per epoch	3.7 s	6.2 s	2.3 s

Table 4-10: Errors and training time in 2D synthetic ultrasound experiments 6, 10 and 11.

The development of the prediction errors over the epochs is shown in Figure 4.15. This figure, along with Table 4-10, shows that reducing the batch size may have improved the accuracy

slightly, but at the cost of a great increase in training time. Increasing the batch size produced larger errors.

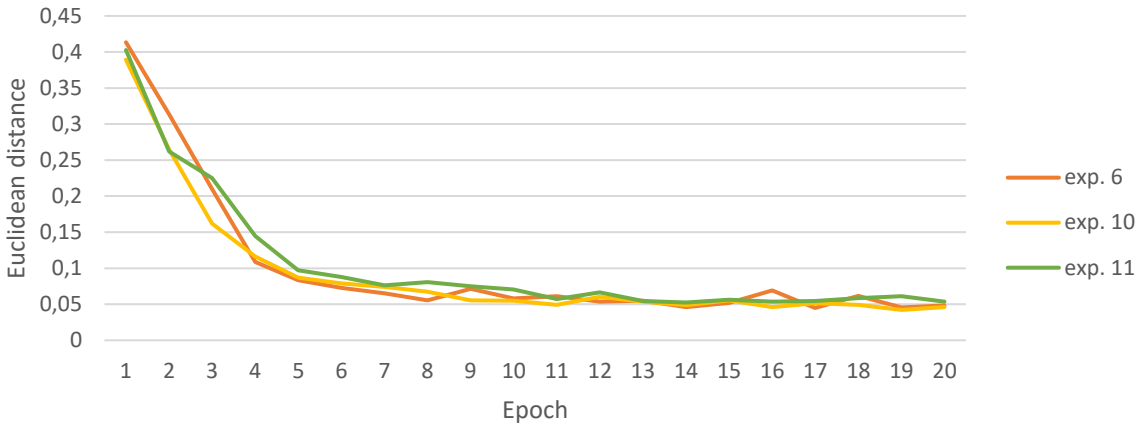

Figure 4.15: Average error in image location prediction - 2D synthetic ultrasound experiments 6, 10 and 11. The batch size in exp. 6, exp. 10, and exp. 11 was 10, 5, and 20, respectively.

4.3.6 Experiments 12 and 13

Experiment 12 (apply L^2) and exp. 13 (increased L^2) tested the effect of applying L2-regularization. The results of these experiments are shown in Table 4-11, along with the results from exp. 6 for comparison.

Measurement	**Exp. 6** L^2-reg. 0	**Exp. 12** L^2-reg. 0.001	**Exp. 13** L^2-reg. 0.01
Average error over five runs after 20th epoch	0.0481	0.0603	0.123
Smallest error obtained	0.0353	0.0507	0.0758
Training time per epoch	3.7 s	4.5 s	4.5 s

Table 4-11: Errors and training time in 2D synthetic ultrasound experiments 6, 12 and 13.

The development of the prediction errors over the epochs is shown in Figure 4.16. Table 4-11 shows that L^2-regularization was not beneficial in these experiments, but Figure 4.16 suggests that similar (or better) results could possibly be achieved by training for many more epochs.

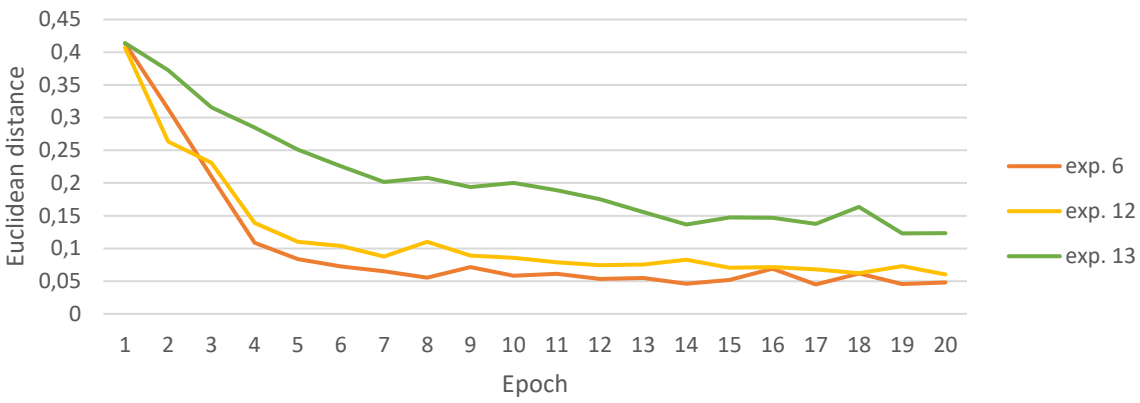

Figure 4.16: Average error in image location prediction - 2D synthetic ultrasound experiments 6, 12 and 13. The L^2-regularization in exp. 6, exp. 12, and exp. 13 was zero, 0.001, and 0.01, respectively

4.3.7 Experiments 14, 15 and 16

Experiment 14 ($yScale_{loss}$ 1), exp. 15 ($yScale_{loss}$ 2) and exp. 16 ($yScale_{loss}$ 4) tested the effect of scaling the y-coordinate during training and testing. Table 4-12 shows the Euclidean distance, the mean absolute error in the x-direction (MAEX) and the mean absolute error in the y-direction (MAEY). These metrics are shown both as the average over five runs after the 40th epoch, and as the best obtained value for any run and any epoch. In all the experiments, the Euclidean distance has been corrected for the assumption that the probe movement was twice as long in the y-direction as in the x-direction, by using $yScale_{test} = 2$, while the MAEY has not been corrected in this manner. Therefore, a Euclidean distance of 1 and a MAEX of 1 both indicate a length similar to the total latitudinal movement range, while a MAEY of 1 indicate a length similar to the total longitudinal movement range. The numbers in parenthesis indicate the relative change compared to exp. 14.

Measurement	**Exp. 14** $yScale_{loss} = 1$	**Exp. 15** $yScale_{loss} = 2$	**Exp. 16** $yScale_{loss} = 4$
Avg. Euclidean dist. over five runs after 40th epoch	0.0736	0.0586 (-20.4%)	0.0778 (+5.71%)
Smallest Euclidean distance	0.0477	0.0468 (-1.89%)	0.0550 (+15.5%)
Avg. MAEX over five runs after 40th epoch	0.0219	0.0190 (-13.2%)	0.0369 (+68.5%)
Smallest MAEX	0.0104	0.0133 (+27.4%)	0.0196 (+88.5%)
Avg. MAEY over five runs after 40th epoch	0.0333	0.0257 (-22.8%)	0.0297 (-10.8%)
Smallest MAEY	0.0214	0.0193 (-9.81%)	0.0206 (-3.74%)

Table 4-12: Errors in 2D synthetic ultrasound experiments 14, 15 and 16.

The table shows that an effect of scaling the y-coordinate is to change the relative priority given to training in the two dimensions, as the smallest MAEY is achieved in exp. 16, and the smallest MAEX is achieved in exp. 14.

The development of the Euclidean distance over the epochs, averaged over five runs per experiment, is shown in Figure 4.17. The order of the three curves after the 40th epoch appears to be random, so emphasis should be placed on the smallest distances obtained at any epoch in Table 4-12

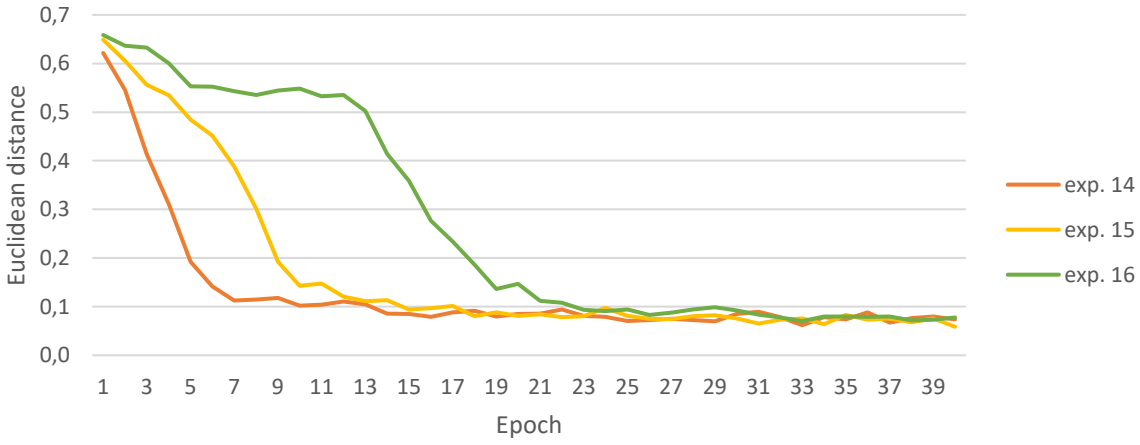

Figure 4.17: Average error in image location prediction - 2D synthetic ultrasound experiments 14, 15 and 16. The $yScale_{loss}$ in exp. 14, exp. 15, and exp. 16 was 1, 2, and 4, respectively.

Table 4-13 shows the average difference between MAEX and MAEY over the epochs, such that a positive number indicates how much greater MAEY was than MAEX. This table shows that increasing the y-coordinate scaling caused a greater relative priority on learning the y-coordinate than the x-coordinate.

Exp. 14 $yScale_{loss} = 1$	**Exp. 15** $yScale_{loss} = 2$	**Exp. 16** $yScale_{loss} = 4$
0.0228	0.0121	-0.00105

Table 4-13: Average differences between MAEX and MAEY in 2D synthetic ultrasound experiments 14, 15 and 16.

The developments of MAEX and MAEY over the epochs are shown in Figure 4.18, Figure 4.19 and Figure 4.20, for exp. 14, 15 and 16, respectively. Again, the results are averaged over five runs. In exp. 14, MAEX clearly scores better than MAEY. This difference is reduced in exp. 15, and not visible in exp. 16, further supporting that scaling the y-coordinate changes the relative priority given to training in the two dimensions.

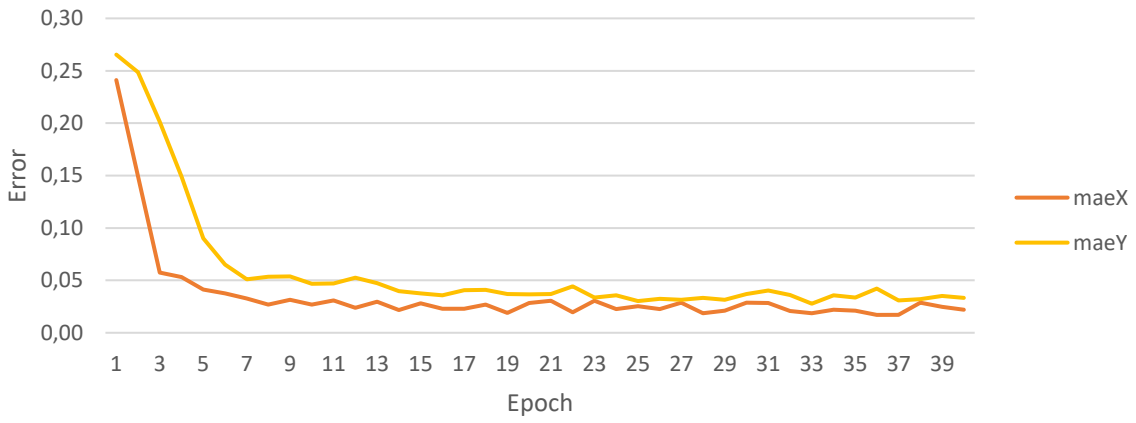

Figure 4.18: Average MAEX and MAEY - 2D synthetic ultrasound experiment 14.

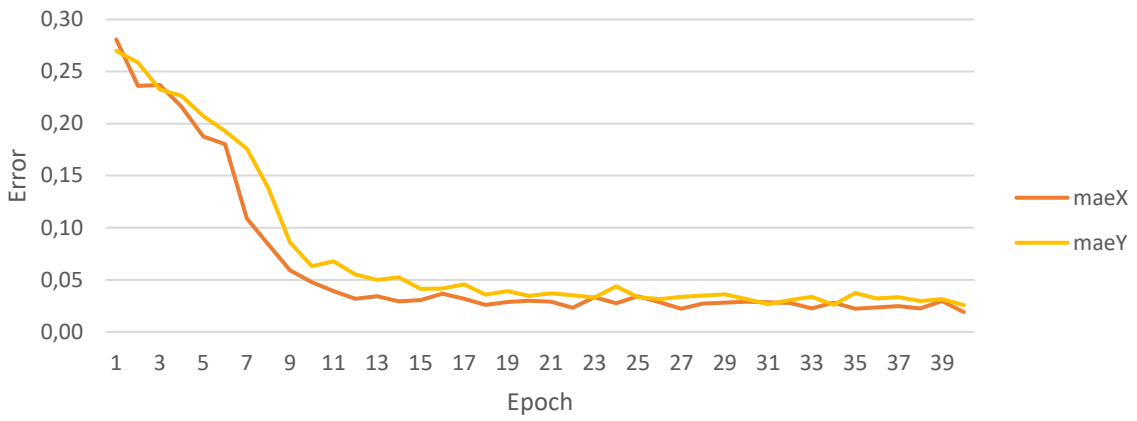

Figure 4.19: Average MAEX and MAEY - 2D synthetic ultrasound experiment 15.

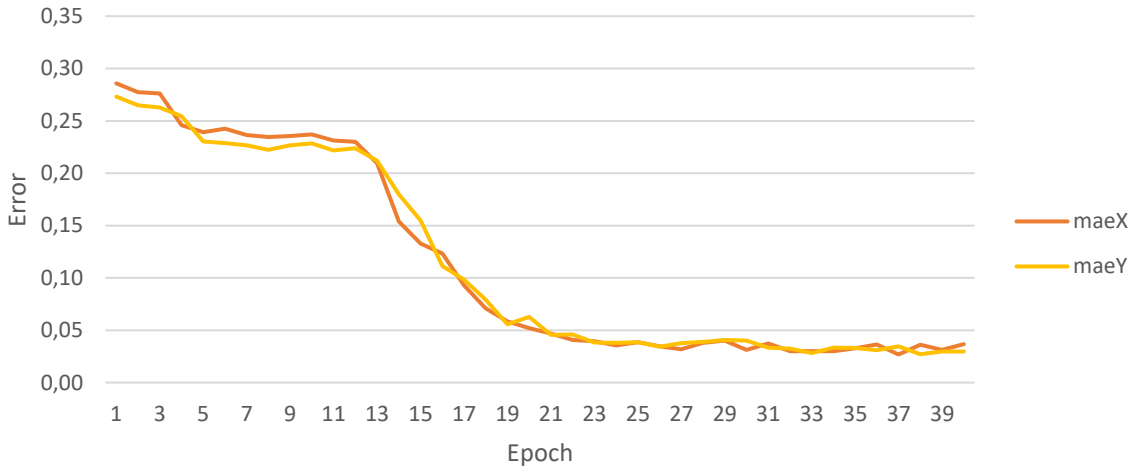

Figure 4.20: Average MAEX and MAEY - 2D synthetic ultrasound experiment 16.

4.4 Lower Arm Dataset

4.4.1 Experiments 1, 2 and 3

The results of exp. 1, 2 and 3 are presented together, because they tested the effect of moving from an AlexNet-like network (exp. 1) to a pre-trained VGG16-like network (exp. 2), and finally to the same pre-trained network with more layers unlocked for training (exp. 3). The results from these experiments are shown in Table 4-14. The smallest errors are obtained with

the pre-trained network of exp. 3, where only the first four layers are frozen. This shows the benefit of using pre-trained networks.

Measurement	**Exp. 1** AlexNet-like	**Exp. 2** VGG16-like Layer 1-5 frozen	**Exp. 3** VGG-like Layer 1-4 frozen
Average MAE over five runs after 20th epoch	0.0418	0.0494	0.0290
Smallest MAE obtained	0.0307	0.0379	0.0266
Training time per epoch	16 *s*	13 *s*	16 *s*

Table 4-14: Errors and training time in lower arm experiments 1, 2 and 3.

The development of the MAE over the epochs, averaged over five runs per experiment, is shown in Figure 4.21.The figure shows the benefit of using pre-trained networks, especially when unlocking layers, so that the start values of the weights can be fine-tuned to the new ultrasound domain.

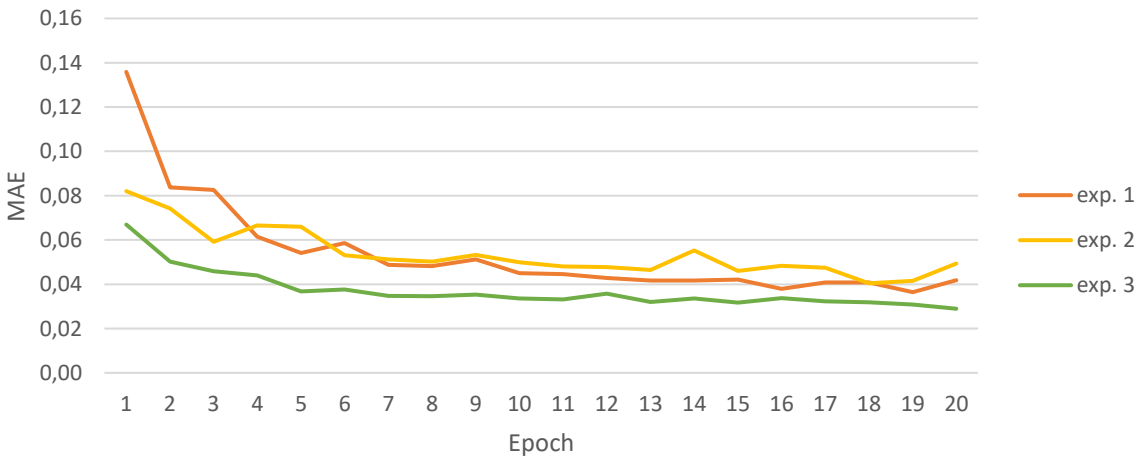

Figure 4.21: Average error in image location prediction - Lower arm experiments 1, 2 and 3. Exp. 1 used an AlexNet-like network, exp. 2 used a pre-trained VGG16-like network with layers 1-5 frozen, and exp. 3 used the same VGG16-like network with layers 1-4 frozen.

4.4.2 Experiment 4

In exp. 4 (1D: cross-validation), cross validation was applied, with one person at a time acting as the test set. The results are presented in Table 4-15.

Test set	**Average MAE over five runs after 20th epoch**	**Smallest MAE obtained**
Person 1	0.0695	0.0557
Person 2	0.0411	0.0324
Person 3	0.204	0.175
Person 4	0.0903	0.0684
Person 5	0.0643	0.0583
Person 6	0.0543	0.0444
Person 7	0.0485	0.0431
Person 8	0.0819	0.0721
Person 9	0.0958	0.0812
Cross validation average	**0.0833**	

Table 4-15: Errors in lower arm experiment 4.

This means that, on average, the network predicted the correct positions with an error of 8.3% of the total recording distance. The development of the MAE over the epochs, for each choice of test person (tp), is shown in Figure 4.22.

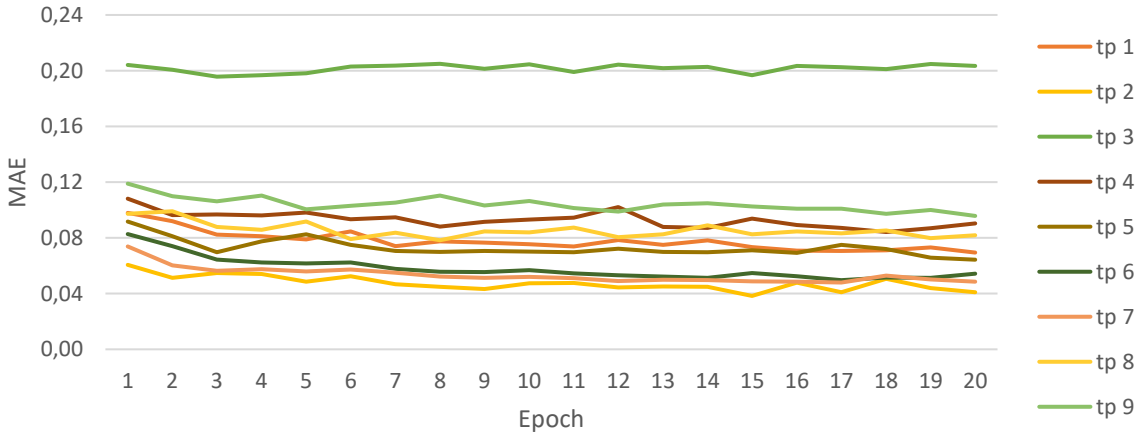

Figure 4.22: Average error in image location prediction - Lower arm experiment 4.

Based on Figure 4.22, testing on person 3 seemed like an outlier, as the error was far greater than that of the other test persons, and no improvement could be seen over the 20 epochs. For the other test persons, the errors were stable, but slightly improving. Figure 4.23 therefore shows the development of the cross-validation MAE with testing on person 3 removed. Here, the average error is clearly reduced over the epochs, and, given more epochs, further improvements are likely.

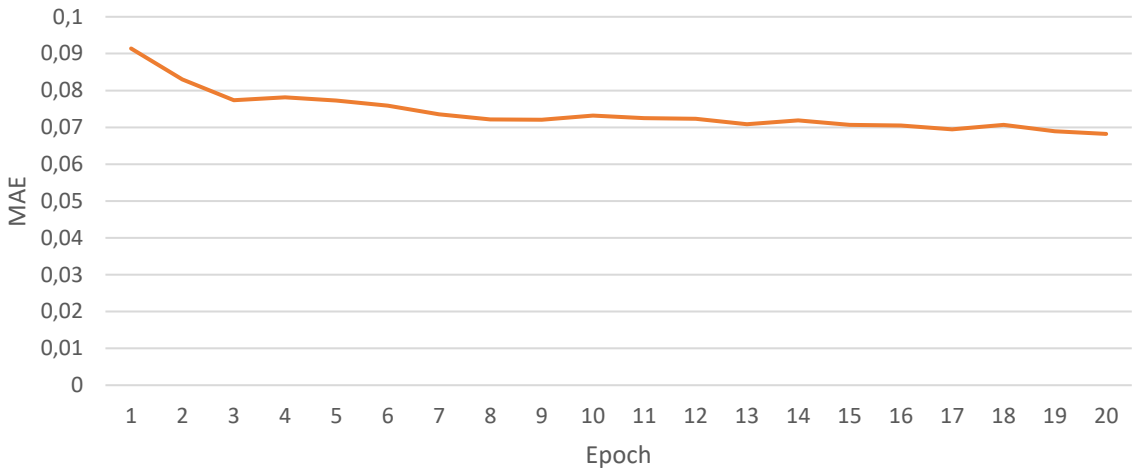

Figure 4.23: Cross validation error in image location prediction - Lower arm experiment 4.

With an average training time of 16 seconds per epoch, the total training time for this experiment was four hours.

4.4.3 Experiment 5

In exp. 5 (2D: train and test on all persons), images from all offsets were used, and the test set consisted of every 10^{th} image from all persons. Table 4-16 shows the Euclidean distance, the mean absolute error in the x-direction (MAEX) and the mean absolute error in the y-direction (MAEY). These metrics are presented both as the average over five runs after the 20^{th} epoch, and as the smallest value obtained in any run at any epoch. The Euclidean distance has been corrected for the difference between the arm's length and width, while MAEX and MAEY show the error in percentage of their respective dimensions' total movement ranges. The result of this is that a Euclidean distance of 1 and a MAEX of 1 corresponds to 12.5 cm (the average

half-circumference of the arm), while a MAEY of 1 corresponds to 24 cm (the average length of the arm).

Measurement	Value
Avg. Euclidean dist. over five runs after 20th epoch	0.0574
Smallest Euclidean distance	0.0540
Avg. MAEX over five runs after 20th epoch	0.0207
Smallest MAEX	0.0175
Avg. MAEY over five runs after 20th epoch	0.0246
Smallest MAEY	0.0231

Table 4-16: Errors in lower arm experiment 5.

The development of the Euclidean distance, averaged over the five runs, is shown in Figure 4.24. Based on this figure, the accuracy could possibly be further improved with more epochs.

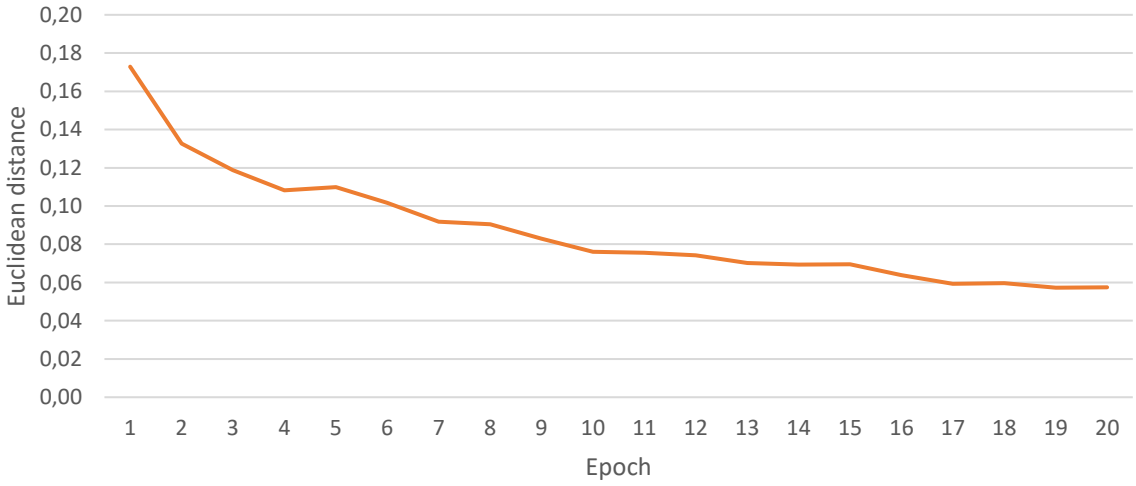

Figure 4.24: Average error in image location prediction - Lower arm experiment 5.

4.4.4 Experiments 6, 7 and 8

In these experiments, images from all offsets were used, but the test set consisted of all images from one person, with cross validation applied. As in exp. 4 (1D: cross-validation), the results when using person 3 as the test set were visibly worse than the rest. However, the difference was not as pronounced as in exp. 4, so the results were included in the cross-validation computation.

The results from exp. 6 (2D: $yScale_{loss}$ 1), exp. 7 (2D: $yScale_{loss}$ 2) and exp. 8 (2D: $yScale_{loss}$ 4) are shown in Table 4-17, which should mostly be interpreted as explained in section 4.4.3. However, the average values presented are over all nine variations of the cross-validation, and over five runs per variation. Also, the smallest values presented are for any test person, any run, and any epoch. There were considerable variations in the results depending on the choice of test person, but the details are not presented here. The table shows, unexpectedly, that the smallest MAEY increased with higher y-coordinate scaling, but, simultaneously, the smallest MAEX increased a lot more. This shows that changing the y-coordinate scaling has an effect the relative priority given to training in the two dimensions.

Measurement	**Exp. 6** $\boldsymbol{yScale_{loss} = 1}$	**Exp. 7** $\boldsymbol{yScale_{loss} = 2}$	**Exp. 8** $\boldsymbol{yScale_{loss} = 4}$
Avg. Euclidean dist. over five runs after 20th epoch	0.197 (2.46 cm)	0.203 (2.54 cm)	0.206 (2.58 cm)
Smallest Euclidean distance	0.133 (1.66 cm)	0.139 (1.74 cm)	0.145 (1.81 cm)
Avg. MAEX over five runs after 20th epoch	0.0920 (1.15 cm)	0.105 (1.31 cm)	0.114 (1.43 cm)
Smallest MAEX	0.0439 (0.549 cm)	0.0592 (0.740 cm)	0.0707 (0.884 cm)
Avg. MAEY over five runs after 20th epoch	0.0727 (1.74 cm)	0.0716 (1.72 cm)	0.0708 (1.70 cm)
Smallest MAEY	0.0428 (1.03 cm)	0.0438 (1.05 cm)	0.0452 (1.08 cm)

Table 4-17: Errors in lower arm experiments 6, 7 and 8.

The development of the Euclidean distance over the epochs, averaged over five runs per experiment, is shown in Figure 4.25. The figure shows that the intra-order of the accuracies of the three experiments was rather stable over the epochs.

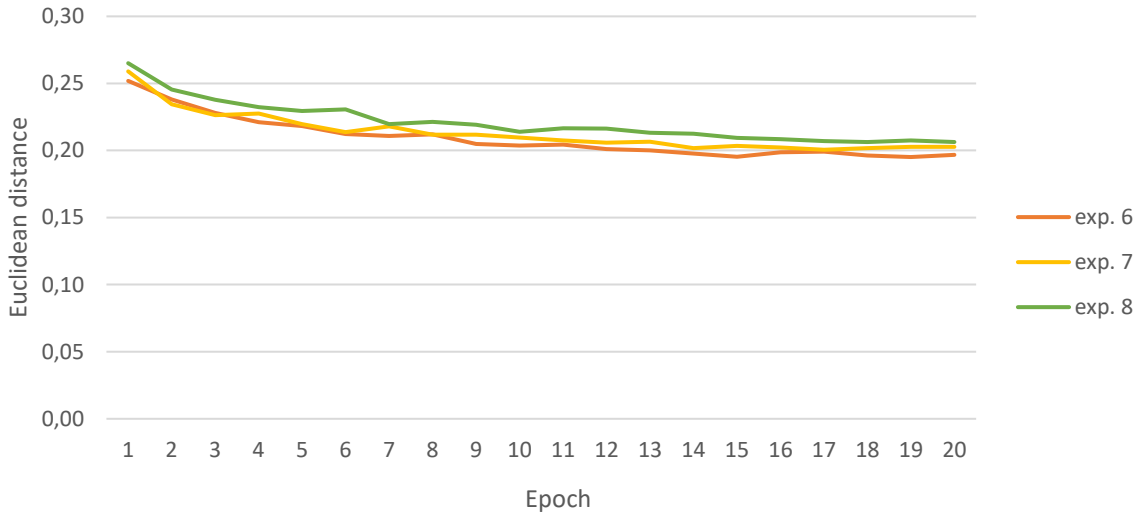

Figure 4.25: Average error in image location prediction - Lower arm experiments 6, 7 and 8. The $yScale_{loss}$ in exp. 6, exp. 7, and exp. 8 was 1, 2, and 4, respectively.

Table 4-18 shows the average difference between MAEX and MAEY over the epochs, such that a negative number indicates how much smaller MAEY was than MAEX. This table shows that with increased y-coordinate scaling more priority was given to learning the y-coordinate than the x-coordinate.

Exp. 6 $\boldsymbol{yScale_{loss} = 1}$	**Exp. 7** $\boldsymbol{yScale_{loss} = 2}$	**Exp. 8** $\boldsymbol{yScale_{loss} = 4}$
-0.0271	-0.0416	-0.0538

Table 4-18: Average difference between MAEX and MAEY in lower arm experiments 6, 7 and 8.

Figure 4.26 and Figure 4.27 show the development of MAEX and MAEY over the epochs for each experiment. The results are averaged over five runs. Note that in Figure 4.27, the second axis starts at 0.06, in order to better highlight the differences between the curves. The MAEX

was clearly increased with increased y-coordinate scaling, but with MAEY, a clear pattern is harder to see, possibly suggesting that order of the experiment results may have been random.

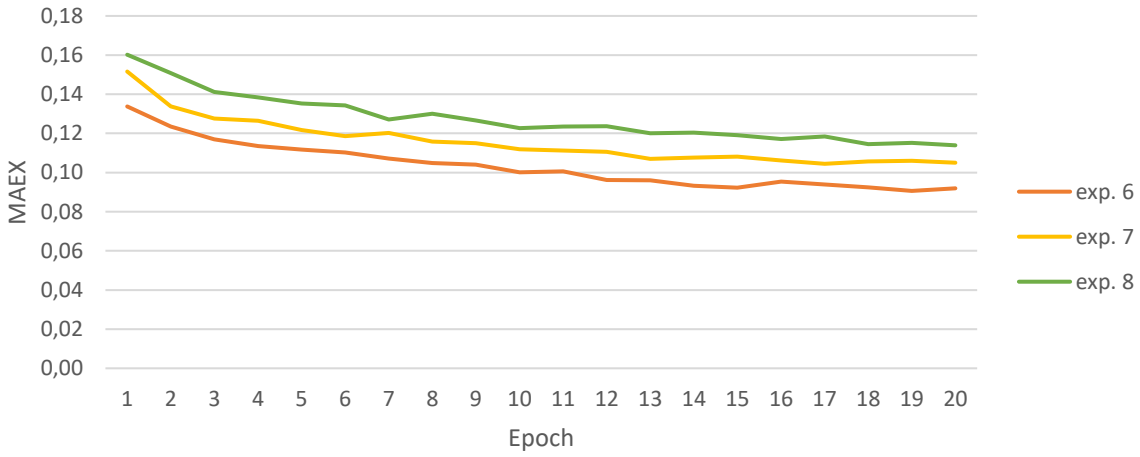

Figure 4.26: Average MAEX - Lower arm experiments 6, 7 and 8. The $yScale_{loss}$ in exp. 6, exp. 7, and exp. 8 was 1, 2, and 4, respectively.

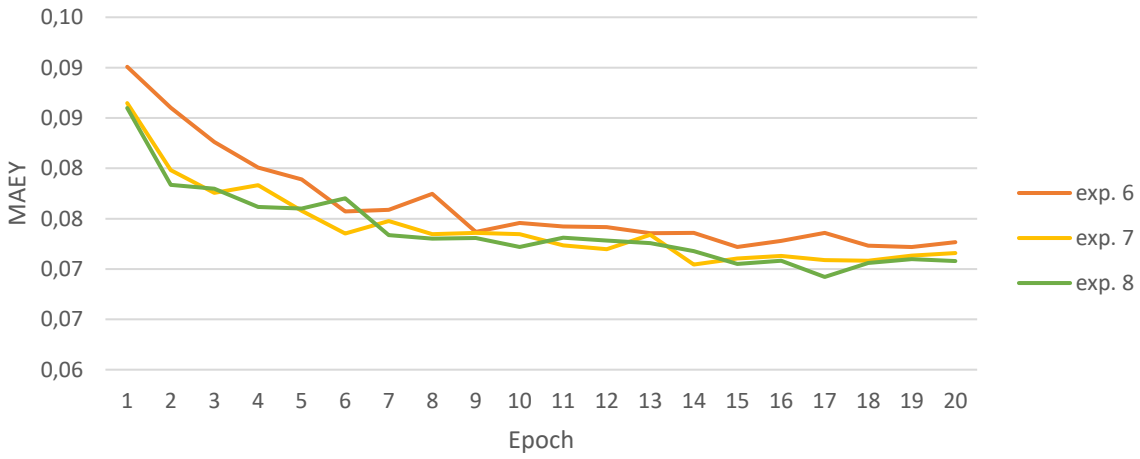

Figure 4.27: Average MAEY - Lower arm experiments 6, 7 and 8. The $yScale_{loss}$ in exp. 6, exp. 7, and exp. 8 was 1, 2, and 4, respectively.

The average training time was 86 seconds per epoch, giving a total cross validation training time of 22 hours for each experiment

4.5 Lower Spine Dataset

The highest classification accuracy achieved for any test person, in any run, and for any epoch, was 90.9% (person 2, run 3, epoch 1). The highest cross-validation accuracy was 66.9%, achieved after the third epoch.

The development of the average classification accuracy over the epochs for each test person is shown in Figure 4.28, along with the cross-validation average accuracy. Apparently, the network was not able to improve the classification accuracy over the epochs in this experiment.

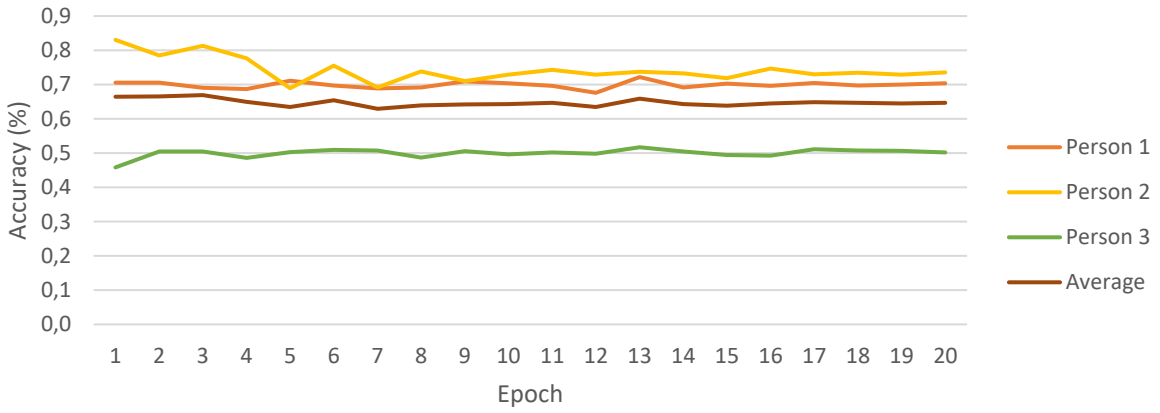

Figure 4.28: Classification accuracy in the lower spine experiment.

5 Discussion

This chapter discusses the results of the experiments, in the light of the goals and research questions. The chapter is divided into separate sections for each dataset.

5.1 Regional Anesthesia Dataset

The average classification accuracy of exp. 1 (cross-validation, no pre-training) after the 10th epoch (73.7%) was similar to the accuracy obtained in Fossan (2016) (74.4%), the difference being that the custom neural network was replaced by the well-known VGG16 network. As seen in Figure 4.1, the accuracy of person 2 was close to 100% after the first epoch, and was then gradually reduced, as opposed to the accuracies of the other persons, which was gradually improved. A good explanation for this behavior was not found, and the pattern was repeated in several control tests which were performed.

The best score, using person 2 as the test set, would be obtained by stopping after the first epoch, while the other persons had their top accuracies at later points, but since these were cross-validation experiments, the performance of the network had to be measured by the average accuracy, which tries to estimate how the network's hyperparameters and architecture would perform on larger training sets. The average accuracy was rather stable over the epochs, but since the results of person 2 were so good in the beginning, the best average accuracy was obtained after the second epoch. Still, if the network had been trained on all four persons and tested on a new population, it is hard to conclude that the best performance most likely would be obtained from only training for two epochs, since the number of persons involved in the experiments was so low.

Just as in Fossan (2016), the results when using person 2 as the test set were a lot higher than the rest. This might seem odd, since all the persons were trained on each other, but a possible explanation is as follows: Let there be three persons, where persons A and B have *many* features in common, persons A and C have *some* features in common, and persons B and C have *few* features in common. First, when using person A as the test set, the network would benefit a lot from training on person B and somewhat from training on person C. Second, when using person B as the test set, the network would benefit a lot from training on person A, but little from training on person C. Third, when using person C as the test set, the network would benefit somewhat from training on person A, but little from training on person B. In this case, the result

would be best using person A as the test set, second best using person B as the test set, and worst using person C as the test set. This arguing could possibly be transferred to the results achieved in these experiments to explain the variation between the test persons.

In exp. 2 (pre-training applied), there was a substantial increase in classification accuracy. The average accuracy after the 10th epoch (85.3%) was more than 10 percentage points higher than the accuracy in exp. 1 (no pre-training), and a lot higher than what a random classification scheme would produce ($\frac{1}{3}$, see section 2.5.1). As there is potential for even further improvements of this accuracy, a positive answer can be given to *RQ1*; neural networks can learn to predict which part of the body an ultrasound image originates from.

Looking at Figure 4.3, it seems clear that the increase in accuracy between exp. 1 and exp. 2 was not a coincidental increase, as the curve of exp. 2 follows the curve of exp. 1 with a rather stable difference. The only change made between exp. 1 and 2 was to load pre-trained weights into the first five layers of the network. These weights were acquired from a network trained on the image-net database, which does not contain any ultrasound images. Since the accuracy in exp. 2 was improved, some basic features must be common to both natural photographs and ultrasound images, which could e.g. be dots, corners, clusters, short lines, etc. In the light of this, an affirmative answer can be given to *RQ4*; pre-training on natural photographs can be applied to the ultrasound domain to help increase the accuracy of the neural network.

In exp. 3 (train and test on all persons), every 10th image in each image sequence was assigned to the test set, and the rest of the images were assigned to the training set, so there was no cross-validation involved. This means that the network could train on images that were very similar to the test set images. If all images from each recording sequence had been utilized, the network would be able to train on almost identical images, acquired from almost the same spot. Since only 100 images were extracted from each recording sequence, spread out evenly over the length of the recording, the situation was not ideal, but still, the similarity between the training images and the test images was far greater than in exp. 2 (training on three, testing on one). This explains the high classification accuracy obtained in exp. 3; 0.996, which corresponds to 239 of 240 test images being classified correctly. Naturally, exp. 3 is artificial, since in a real application, a network would not be trained on the same persons as it would be tested on. Nevertheless, the results show that, given more similar images to train on, the classification accuracy can be greatly improved. In a real application, this could translate to training on

thousands of persons, increasing the probability that the anatomies of some of the training persons are very similar to a test person.

In exp. 4 (remove person 3), the removal of the weakest scoring person clearly improved the results of exp. 1. Each of the persons 1, 2 and 4 scored better, as seen from comparing Table 4-1 with Table 4-3. The average cross-validation accuracy at the end of the 10^{th} epoch increased by eight percentage points – from 0.737 to 0.817, but the tendency was present during all the epochs, as seen from Figure 4.6, where the cross-validation accuracies of exp. 1 and 4 (with and without person 3) are compared. Since the remaining test persons benefited from not having to train on the excluded person 3, the images from the recordings on person 3 probably differed a lot from the other recordings. However, the reason for this is not obvious. One explanation could be that, by accident, the technical aspects of the recordings differed, such as the gain setting, the tilting, or the rotation of the ultrasound probe. However, since consistency was a major focus during the recordings, a more likely explanation is that the anatomy of person 3 differed a lot from the others. Whether this was arbitrary, or a result of person 3 being the only woman in the experiment, is difficult to say due to the small number of test persons. To show whether gender plays an important role when training networks in this manner, a new study with far more test persons would be necessary.

In summary, the results of the regional anesthesia experiments show that pre-training on natural photographs can be applied to the ultrasound domain to speed up the learning process substantially, and that neural networks can be used to approximately determine where on the body an ultrasound image is acquired from. This is not directly applicable towards guiding the ultrasound user closer to a specified body location (*RQ2*), but it shows that it is possible to make a system aware of approximately were on the body the probe is located (*RQ1*). The experiments could be expanded to include many more regions, possibly covering the whole body. If these regions could be rather small while maintaining a high classification accuracy, some sort of guidance could possibly be produced by the system, but probably not of high enough accuracy to be of use in a real application. A better approach would probably be to use the method of the regional anesthesia experiments to classify the approximate image location, and then let a more accurate system take over the guidance.

5.2 1D Synthetic Ultrasound Dataset

At its best, the network used in the 1D synthetic experiments predicted the location of the synthetic ultrasound images with an error of less than 2% of the total recording distance, which is well below the error of $\frac{1}{3}$ that a random placement would produce (section 2.5.2). This was a rather small error, as could be expected from training on synthetic images of simple structures with smooth, uniform changes over distance. Also, since every second image in the recording sequence were assigned to the training set, the network could train on images very similar to those in the test set. As can be seen from Figure 4.7, most of the learning took place early, and the network did not benefit much from training beyond epoch 30.

The effect of the data augmentation (exp. 2) was different than expected; As can be seen in Figure 4.9 data augmentation did not reduce the error, but rather made the predictions worse during the first 20 epochs. After that, the results from the two experiments evened out. A possible explanation for this is that both the training images and the test images were recorded along the same perfectly straight line, so the original training images were the optimal images to train on, producing a steep learning curve. On the other hand, exposing the network to shifted images disturbed the ideal situation, and caused the network to spend more time to learn the correct mapping function, as visible from the shallower learning curve in Figure 4.9. However, after some time, the network learned to overcome this disturbance, and since this network ended up with about the same MAE as the first network, it is possibly a more robust network in terms of generalization. While the first network could rely on memorizing the training images, which were very similar to the test images, the second network could not do this to the same extent. Therefore, although it did not reduce the error in these experiments, using data augmentation techniques, like shifting, might be beneficial when training on ultrasound data from multiple recordings and/or people.

Despite the small size of the 1D synthetic experiments, and the relatively easy task given to the neural network, the results point to the possibility of using neural network regression training to predict ultrasound image locations. This is a possible answer to *RQ2*; If a neural network can be trained to predict the location of ultrasound images using regression training, then guidance to a desired coordinate can be given. Certainly, there is a large difference between a simple, synthetic experiment in one dimension, and a real ultrasound application, but at least the results do not dismiss the proposed method. However, the error of location prediction would probably

be much higher if the method used in these experiments had been applied to real ultrasound data, as was the case in the lower arm experiments.

Another thing to note is that the labelling of the ultrasound images does not have to be linear, even though the recording is performed linearly. For example, the first and the last image in a linear recording could be labelled 0 and 1, respectively, while the mid-coordinate, 0.5, could be assigned to the image in the sequence that contain some particular feature, like the point where a certain blood vein splits in two. This splitting could occur at slightly different places from person to person, and so would consequently the position of the 0.5 label. The rest of the labels could be interpolated between 0 and 0.5, and between 0.5 and 1. To find the vein split, the probe user could move the probe in *one* direction if the predicted label of the current probe position was smaller than 0.5, and in the *other* direction if the current label was greater than 0.5.

5.3 2D Synthetic Ultrasound Dataset

5.3.1 Experiments 1 and 2

Moving from network 1 (simple, custom network in exp. 1) to the much larger network 2 (AlexNet-like network in exp. 2), drastically reduced the prediction error, as seen in Table 4-6 and Figure 4.10. Network 2 had a much steeper learning curve during the first 11 epochs, after which both learning curves were rather shallow. Looking at Figure 4.10, it is possible that smaller errors could have been obtained from increasing the number of epochs, but these experiments showed that network 2 was superior to network 1, and consequently the next experiments were based on network 2.

The smallest error obtained in the second experiment, 0.0639, was not as low as the error obtained in the 1D synthetic experiments, as might be expected since the prediction task was more complex (2 dimensions instead of 1 dimension). Since the Euclidean distance with scaling of the y-coordinate was used as the test function, the error can be interpreted as the average distance between the predicted location and the correct location, in percentage of the horizontal range of the probe, since this axis was not scaled – in this case about 6.4% of the horizontal range of the probe.

As expected, the training time increased a lot when moving from network 1 to network 2; from 0.6 s per epoch to 9.6 s per epoch. The subsequent experiments tried to lower this training time while maintaining or reducing the prediction error. Lowering the training time was deemed important, since it was anticipated that the size of the lower arm dataset would require a lot of time for training.

5.3.2 Experiments 3, 4 and 5

The simplifications done in exp. 3 (remove layer 7), exp. 4 (remove layer 4), and exp. 5(reduce conv. kernels) lowered the training time while maintaining approximately the same level of accuracy as in exp. 2 (AlexNet).

As seen in Table 4-7, dropping a fully connected layer (exp. 3) produced the smallest prediction error of the four compared experiments. Dropping a whole layer reduces the capacity of a neural network and can make the network unable to solve complex tasks (Goodfellow et al., 2016, p. 112), but at the same time, having unnecessarily many nodes can increase the time needed to learn the same function, and also may increase the risk of overfitting. Figure 4.11 shows that exp. 3, without the extra fully connected layer, learned quicker during the first epochs, but also that exp. 2 achieved almost the same prediction error, given more epochs. It is therefore possible that exp. 2 could have achieved the same, or lower error as exp. 3, if even more epochs had been allowed.

As seen in Figure 4.11, the steep learning curve of exp. 3 became somewhat less steep in exp. 4 and 5. Also, these two experiments did not achieve the same prediction error as exp. 3, but the great reduction in training time compared to the small increase in error caused further experiments to be based on exp. 5.

5.3.3 Experiments 6 and 7

Adjusting the learning rate λ had a large impact on the prediction error. As can be seen in Table 4-8 and Figure 4.12, increasing λ from 10^{-5} to 10^{-4} (exp. 6) produced a steep learning curve during the first epochs, and also reduced the error measured at the end of the 20^{th} epoch. Still, after the initial steep learning, the curve behaved rather unstable, and it was believed that the use of decay could smooth out the learning, possibly achieving even better precision (This was

attempted in exp. 8 and 9). The smallest error produced during the sixth experiment, 0.0353, was substantially lower than the error of 0.521 that a random positioning would produce (see section 2.5.3).

Raising λ even further (exp. 7) produced worse results than exp. 5. As explained in section 2.1.5.3, raising λ too much can cause the learning process to overshoot the optimal weight values, and make it hard to get close to any good solutions in the model's hypothesis space. This seems to be the case in exp. 7 when looking at Figure 4.13, where the performances of the individual runs are very varied, and especially one of the runs struggles with finding a good solution. Further experiments were consequently based on exp. 6.

5.3.4 Experiments 8 and 9

The introduction of decay in exp. 8 (apply decay) and exp. 9 (increased decay) worked as expected, in that it made the learning curve smoother towards the end of the epochs. In Figure 4.14, this is most visible for exp. 9, which applied the most decay. However, the main goal of applying decay was to reduce the prediction error, and this was not achieved. Although the average error over five runs after the 20^{th} epoch was slightly better in exp. 9 compared to exp. 6 (0.0461 vs. 0.0481), the smallest error obtained in any run and any epoch was slightly better in exp. 6 (0.0353 vs. 0.0358). Both of these differences were too small to be considered significant. The use of decay was therefore not continued.

5.3.5 Experiments 10 and 11

Dividing the batch size by two (exp. 10) increased the training time per epoch by a factor of 1.68. However, this increase in training time can not be seen isolated from the other effect of halving the batch size – doubling the number of weight updates per epoch. The combined effect is an increase in the number of updates per second by a factor of $\frac{2}{1.68} \approx 1.19$. Still, as described in section 2.1.3.2, with a smaller batch size, each batch becomes a poorer approximation of the whole training set, and consequently each update may be less accurate. Looking at Figure 4.15, it seems like decreasing the batch size did not improve the average results in this case, although Table 4-10 shows that the smallest prediction error obtained in any run and any epoch was improved somewhat. Similarly, doubling the batch size reduced the number of updates per

second by a factor of $\frac{3.7/2.3}{2} \approx 0.81$, while each update may have been more accurate. Looking at Table 4-10, the effect of doubling the batch size (exp. 11) was an increase of both the average and the smallest errors. In summary, it was not found beneficial to either increase or decrease the batch size, and further experiments were therefore based on exp. 6.

5.3.6 Experiments 12 and 13

The application of L2-regularization in exp. 12 (apply L^2) and exp. 13 (increased L^2) caused a small increase in the training time, probably because of increased complexity in the loss function. At the same time, both the average and the smallest prediction error of the network increased, as seen in Table 4-11– somewhat in exp. 12 and even more in exp. 13. Still, looking at Figure 4.16, the error of exp. 6 did not improve much during the last 10 epochs, while the error of the experiments with L2-regularization did improve during this interval. It is therefore not unlikely that the experiments with L2-regularization could have produced the same or smaller error as exp. 6, given many more epochs. However, for this small study, the further use of L2-regularization was discarded.

5.3.7 Experiments 14, 15 and 16

The results from these experiments are not directly comparable to the earlier experiments, since the test function now incorporates a scaling of the y-coordinate to compensate for the assumption that the total movement range was twice as long in the longitudinal direction as in the latitudinal direction. The errors of this experiment can be compared to the second derivation of the random positioning error in section 2.5.3, since one of the axis is of length 2. As seen in Table 4-12, the smallest Euclidean distance error of 0.0468 was achieved in exp. 15 and this error is substantially smaller than the error of 0.805 that a random positioning would produce.

As seen in Table 4-12, moving from exp. 14 ($yScale_{loss}$ 1) to exp. 15 ($yScale_{loss}$ 2) greatly reduced the error (the Euclidean distance). This was expected since exp. 15 scaled the y-coordinate by the same amount in both the loss function and the test function. It was also expected that a further increase in $yScale_{loss}$ to 4 (exp. 16), causing a higher priority on training the y-coordinate than the x-coordinate, would result in an increase of the Euclidean distance compared to exp. 15. This was the case, and as seen in Table 4-12, the Euclidean distance was even higher than in exp. 14 – an effect that was not anticipated. Exp. 14 and 16 were expected

to produce about the same result, since the factor of scaling error was the same, only in different directions; with $yScale_{loss} = 1$ the scaling was half as large as it should be, while with $yScale_{loss} = 4$ the scaling was twice as large as it should be.

When looking at the two dimensions individually, MAEX and MAEY, some of the results were as expected, while others were not. Increasing the $yScale_{loss}$ to 2 produced a higher (and correct) emphasis on learning the correct y-coordinate, so MAEY was improved in exp. 15. However, raising $yScale_{loss}$ to 4 did not further improve the MAEY, although it was still better than exp. 14. In contrast, the best MAEX in exp. 15 was worse than in exp. 14, and even worse in exp. 16, as expected with an increased emphasis on the y-coordinate. The average MAEX was also a lot worse in exp. 16, but, surprisingly, it improved in exp. 15. No reason for this could be found, and the numbers might be coincidental, given the small number of runs, the unsmooth learning curves in all runs and the arbitrary choice of sampling the prediction error at the end of the 40th epoch. As of such, the best MAEX found in any run and any epoch could be a better measurement of the performance.

Figure 4.17 suggests that increasing $yScale_{loss}$ prolonged the time needed to achieve the approximate same Euclidean distance error. It also shows that, although the relative change in the average Euclidean distance at the 40th epoch was large between exp. 14, 15 and 16 (Table 4-12), the order of the three learning curves varied a lot during the last 20 epochs, and the order at the 40th epoch might be random. Therefore, when reading the Euclidean distances in Table 4-12, more emphasis should be placed on the row displaying the smallest Euclidean distance found at any epoch and any run.

Another interesting comparison of the effects on the MAEX and the MAEY of changing $yScale_{loss}$, is found in Table 4-13. This table shows that in exp. 14, the MAEY was 0.0228 higher than MAEX, averaged over all runs and over all epochs. In exp. 14, the x- and y-coordinate were emphasized equally during training, and the result in Table 4-13 therefore shows that the x-coordinate was easier to train than the y-coordinate. Next, when increasing $yScale_{loss}$ to 2, the average difference between MAEX and MAEY is halved, and with $yScale_{loss} = 4$, MAEY is smaller than MAEX on average. This again shows that changing the scaling of the y-coordinate during training has an impact on the priority given to the two dimensions.

The same effect can be shown by looking at Figure 4.18, Figure 4.19 and Figure 4.20. In the first figure, displaying exp. 14, the MAEX is clearly smaller than the MAEY in all epochs. This difference becomes less evident in the second figure, which shows exp. 15, and disappears in exp. 16 in the last figure.

In summary, the effect of scaling the y-coordinate during training seems to be to change the relative priority given to training in the two dimensions. Increasing the scaling above the natural setting puts more emphasis on learning the y-coordinate, but at the risk of increasing the Euclidean distance error.

5.4 Lower Arm Dataset

5.4.1 Experiments 1, 2 and 3

Table 4-14 shows that exp. 1 (1D: no pre-training), based on the best hyperparameters found in the 2D synthetic experiments, performed better than the pre-trained network of exp. 2. However, when unlocking the last convolutional layers in exp. 3, the accuracy surpassed exp. 1 by a large margin, clearly indicating the effectiveness of using pre-trained networks on the ultrasound domain. Also, exp. 1 and 3 had the same training time per epoch, even though the size of the images input to the pre-trained network was almost eight times greater $\left(\frac{224 \times 224}{100 \times 64} \approx 7.8\right)$. This equality of training time was possible since four of the layers in exp. 3 were frozen from training. It is possible that some of the error reduction in exp. 3 can be credited to the size of the input images, so that if this network had been fed images of the same size as in exp. 1, the error may have become larger. This hypothesis was not tested.

At its best, the network in exp. 1 predicted the y-coordinate of all images with an average error of about 3% of the total recording distance. This corresponds to about $0.7\ cm$. Similarly, the network in exp. 2 predicted the image locations with an average error of $0.9\ cm$ at its best, and the network in exp. 3 predicted the image locations with an average error of $0.6\ cm$ at its best. This was more accurate than expected, but as every 10^{th} image was selected for the test set, the network was allowed to train on images very similar to the test set.

In Figure 4.21, the benefit of using pre-trained networks is evident. Exp. 2 and 3 get a head start in the training process since they are pre-trained, and have a much smaller error than exp. 1

after the first epoch. Also, the positive effect of unlocking one more of the frozen layers for training is clearly visible, as the learning curve of exp. 3 is well below the learning curve of exp. 2 in all epochs. Unlocking the whole model would probably produce even better results, especially if given more epochs, but that would simultaneously cause a great increase in the training time, and this was therefore not attempted.

As described in the 2D synthetic experiments, the use of data augmentation could have had a positive effect on the regularization of the networks, and may have reduced the error somewhat. However, since the networks in these experiments took a long time to train – especially in the last experiments – testing data augmentation was not prioritized.

5.4.2 Experiment 4

As expected, in exp. 4 (1D: cross-validation) the prediction error increased when selecting all images from one person as the test set, in a person-wise cross-validation scheme. On average over all test persons, after the 20^{th} epoch, the network was able to predict the image locations with an error of about 8.3% of the total length of the recording, which corresponds to about $2\ cm$. There were, however, large variations depending on which person was used as the test set. The best result in a single run for any test person, was achieved while testing on person 2: About 3.2% error on average, or $0.8\ cm$, which is quite close to the best result achieved in exp. 3. Conversely, while testing on person 3, the smallest error achieved in a single run was 17.5%, or $4.2\ cm$.

As noted in section 4.4.2, person 3 might be considered an outlier, since it deviates considerably from the other persons in Figure 4.22. If testing on person 3 is removed from the experiment, the cross validation average drops to 6.8%, or $1.6\ cm$.

Figure 4.23 shows a rather smooth learning curve, with improvements in most of the epochs. Based on this, the error could probably have been reduced even more with additional epochs.

5.4.3 Experiment 5

The results in exp. 5 (2D: train and test on all persons) were better than expected, but again, the fact that the network was allowed to train on images that were very similar to the test set was probably a major factor.

Looking at Table 4-16, at the end of the 20th epoch the Euclidean distance between the predicted image location and the correct location was on average about $5.7\% \; x \; 12.5 \; cm \approx 0.72 \; cm$. Likewise, the average error in the x-direction after the 20th epoch was about $2.1\% \; x \; 12.5 \; cm \approx 0.26 \; cm$, and the average error in the y-direction after the 20th epoch was $2.5\% \; x \; 24 \; cm \approx 0.60 \; cm$. The best results obtained at any individual run and after any epoch, were slightly lower. As in the 2D synthetic experiments, it seems like the x-coordinate was easier to learn than the y-coordinate. Note that since the half-circumference is much larger near the elbow than at the wrist, the presented errors in the x-direction have a lot of uncertainty, since they assume a uniform half-circumference of $12.6 \; cm$.

When looking at the shape of the learning curve in Figure 4.24, it is also possible that this experiment could have produced lower errors given more epochs.

5.4.4 Experiments 6, 7 and 8

The smallest cross-validation average prediction error achieved in these experiments was $2.46 \; cm$, and the smallest error achieved at any run, for any test person, and at any epoch, was $1.66 \; cm$, or 0.133 before the conversion to centimeters. As these experiments were performed on real ultrasound images in two dimensions, and without training and testing on the same persons, the precision of $1.66 \; cm$ is the best answer to *RQ3* found in this thesis. Although this error is larger than in the 2D synthetic experiments, the error is still smaller than the error of 0.805 that a random positioning would produce, as shown in the second derivation in section 2.5.3. The results provide a possible answer to *RQ2*: Neural networks with regression can be trained to predict the current two-dimensional coordinate of the ultrasound probe, which in turn can be used to guide the probe user to a specified coordinate. Still, the precision achieved in these experiments was too low to give precise guidance in such a system.

The smallest error was unexpectedly obtained during exp. 6, where $yScale_{loss} = 1$. Since $yScale_{test} = 2$ in all these three experiments, the smallest error was expected to be found in exp. 7, where $yScale_{loss} = yScale_{test} = 2$. The pattern was the same for both the cross-validation average Euclidean distance after the 20th epoch and the best Euclidean distance found for any test person, any epoch, and any run; Exp. 6 (2D: $yScale_{loss}$ 1) performed the best, followed by exp. 7 (2D: $yScale_{loss}$ 2) and lastly exp. 8 (2D: $yScale_{loss}$ 4).

A possible reason for this pattern may be found when looking at the results for the individual dimensions. Relative to MAEY, MAEX was prioritized the most in exp. 6 ($yScale_{loss} = 1$), and the least in exp. 8 ($yScale_{loss} = 4$). It was therefore expected to find the smallest MAEX in exp. 6, and the largest MAEX in exp. 8. The results in Table 4-17 confirm this expectation with a 14.1% increase in the average MAEX from exp. 6 to exp. 7, and a 23.8% increase from exp. 6 to exp. 8. On the other hand, with MAEY the results do not match the expectations to the same extend. Since, relative to MAEX, MAEY was prioritized the most in exp. 8 and the least in exp. 6, it was expected that the worst MAEY would be found in exp. 6, and the best MAEY would be found in exp. 8. This was the case, but only with a 1.51% decrease in the average MAEY from exp. 6 to exp. 7, and a 2.61% decrease from exp. 6 to exp. 8. It seems therefore that it was hard to improve the result of the MAEY found in exp. 6, even when a lot more emphasis was put on this dimension. This is a possible reason why the Euclidean distance scored best in exp. 6, as the Euclidean distance is a combination of the MAEX and the MAEY.

It should be noted that, although the first numbers in each cell of Table 4-17 for MAEX and MAEY indicate that the network was better at learning the correct y-coordinate, the opposite is true for the metric distance. This is because the normalized numbers represent the distance as a fraction of the total movement range in the two dimensions. Since the total movement range was about twice as long in the longitudinal direction as in the latitudinal direction, these numbers are misleading when it comes to the real distance, and the converted metric numbers (in parenthesis in Table 4-17) should be compared instead. Doing this, the network was clearly better at learning the x-coordinate than learning the y-coordinate.

A possible interpretation of the metric Euclidean distance error is to imagine a circle with a radius equal to the distance error around the correct location for a certain image. In this case, the best cross-validation average Euclidean distance achieved was $2.46\ cm$, so on average, the prediction of the images in the test set can be imagined placed somewhere on a circle of radius $2.46\ cm$ around the correct image location. This precision is probably far too coarse to be of use in any real medical application. Still, the results are a lot better than a random guess at the location. As shown in section 2.5.3, the expected distance between two random points in a square with unit sides is 0.521. On the other hand, the best result achieved was $2.46\ cm$, which was derived from the number 0.197 – the Euclidean distance as a fraction of the total latitudinal movement range. Since the labels were given values between 0 and 1, in both dimensions, the

numbers 0.197 and 0.521 are comparable, and show a significant improvement over pure guessing.

Figure 4.25 shows that the order of the Euclidean distance results of the three experiments was rather stable over all the epochs; Exp. 6 performed the best, followed by exp. 7 and 8. Also, this figure suggests that little would be gained from allowing more epochs for training, as the curves are almost flat for the last five epochs.

The effect on the difference between MAEX and MAEY from varying $yScale_{loss}$ is perhaps more visible in Table 4-18. The table shows that, on average over all the epochs, and over all variations of the cross-validation, MAEY was 0.0271 lower than MAEX in exp. 6. When doubling $yScale_{loss}$, this difference is increased to 0.0416, and the final doubling of $yScale_{loss}$ increases the difference to 0.0538. This illustrates how more emphasis was put on learning the correct y-coordinate than the x-coordinate, but as described above, the increasing difference was probably more due to decline of the x-coordinate prediction than to improvement of the y-coordinate prediction.

Figure 4.26 shows how the increase in $yScale_{loss}$ caused an increase in MAEX at all epochs, and that this increase was not random, since the curves are of the same shape and transposed in the vertical direction. On the other hand, the same order is not present in Figure 4.27, showing the development of the average MAEY over the epochs. These curves are less smooth and the shapes are more varied than in Figure 4.26. Although it seems clear that exp. 7 and 8 performed better than exp. 6, the intra-order of exp. 7 and 8 seems random, at least between the 5th and the 15th epoch. The order is preserved during the last five epochs, producing the results discussed above, but it seems possible that a new execution of these experiments could have produced a different ordering of the MAEY results at the 20th epoch. Another interesting aspect of Figure 4.27 is that the learning curves of exp. 7 and 8 are steeper than the learning curve of exp. 6 during the first two epochs. This might be due to the increased emphasis on learning the y-coordinate.

5.5 Lower Spine Dataset

As the best cross-validation accuracy achieved was only 66.9%, using the neural network of this experiment in its current form is not a reliable way to determine the presence of a lumbar vertebra. As seen in Figure 4.28, the network was not able to improve the classification accuracy

over the epochs, so the accuracy at the end of the 20th epoch was very close to the accuracy at the end of the first epoch.

One possible reason for the low accuracy obtained is the small number of persons involved in the training. Ultrasound images are noisy, and can appear very different from image to image within a recording, and from person to person. Therefore, the network should ideally have been exposed to lower spine images from hundreds of persons to be able to disregard the noise and learn the important features that distinguish vertebra images from non-vertebra images.

Another problem with this experiment was the manual classification of the images prior to training and testing. The classification was done by the author, who did not have much experience in interpreting lower spine images. To classify the images, the recordings were reviewed at slow speed, and the boundaries between the vertebrae and the gaps was searched for. Although many images were classified with confidence, the boundaries between the regions were hard to determine exactly. Because of the uncertainty, some of the vertebrae and gaps were not included in the experiment at all. Ideally, the manual classification should have been performed by experienced medical personnel.

The dataset used in the experiment was not balanced. 62% of the examples were vertebra images, 24% were gap images, and 14% were sacrum images. As explained in section 2.1.1.5, this can be problematic as it can bias the network towards predicting the vertebra label. Ideally, to remove this bias, images from the three categories should have been randomly selected for training with equal probability. Another imbalance in the experiment was the number of images from each person. The way the experiment was conducted, the network was somewhat more exposed to images from person 2 (1074 images) than from person 1 (816) and from person 3 (972). Although this difference was rather small, in principle it could bias the network towards relying more on the features learned from person 2 than those learned from the others.

The best cross-validation average, 66.9%, was significantly better than what a random classification would give; as explained in section 2.5.1, this would give an accuracy of $\frac{1}{3}$. However, some of this improvement might be the result of a possible bias in the network towards predicting the most frequently seen class during training. Still, if the network had been biased to the point where all images had been classified as vertebrae, the classification accuracy would not be higher than about 62% (the ratio of vertebra images), so the network must have learned something apart from this bias. Ideally, the number of images from each class that were

labelled correctly should have been recorded during this experiment, to be able to further investigate this issue.

In summary, the network of this experiment learned something about the difference between vertebra, gap, and sacrum images. Although the accuracy was far too low to be of any practical use, the proposed method cannot be completely dismissed, as several possibilities for improvement were identified.

6 Conclusion

In this thesis, the possibility of creating an ultrasound guidance system using neural networks has been examined. In order to investigate the potential, various experiments related to five different datasets have been performed. The first set of experiments, performed on the regional anesthesia dataset, showed that neural networks has the potential to classify the approximate body location of ultrasound images. In this way, neural networks can be used to guide the ultrasound probe user closer to a specified body location, by first classifying the approximate location of the current ultrasound image, and then suggesting a probe movement direction towards the specified location.

The accuracy achieved in the first regional anesthesia experiment, 73.7%, was further increased to 85.3% when applying pre-trained weights from another network trained on natural images in the image-net database. This shows that applying pre-training on natural photographs to the ultrasound domain can help speed up training and increase the accuracy of the neural network.

In the 1D and 2D synthetic experiments, the use of neural networks to predict the location of images through regression was demonstrated. As the networks predicted coordinates for the ultrasound images, this method has the potential to provide precise guidance to the probe user. The average error of the predicted locations in these experiments was small, but since the images were synthetic, a lower degree of error could be expected than from real ultrasound images.

During the 2D synthetic experiment, it was shown how tuning of the architecture and hyperparameters could both reduce the prediction error and the required training time in this case. It was also demonstrated how the inclusion of a variable scaling factor in the loss and test functions could compensate for the mismatch between the normalized coordinates used in the image labels and the actual image coordinates. In this synthetic case, the y-axis range was assumed to be twice as long as the x-axis range, and the normalized image label coordinates were compensated for accordingly, by setting the test scaling factor to 2. Setting the loss scaling factor to any other value than this changed the network's relative priority given to training in the two dimensions. However, another effect of this change was that the combined Euclidean distance error was increased.

The experiments performed on the lower arm dataset again showed the effectiveness of applying pre-training on the ultrasound domain. With pre-training applied, the location error was as low as $0.6\ cm$ in the one-dimensional case, when training on nine of ten images from all persons. In the more realistic experiment where training was performed on all persons except for one test person, the location error ranged between $0.78\ cm$ and $4.2\ cm$, depending on the choice of test person. In the two-dimensional case, the average location error on the lower arm dataset was $0.72\ cm$ when training on nine of ten images from all persons. This error was increased to $2.46\ cm$ in the cross-validation scheme ($1.66\ cm$ in the best case).

Although the errors in these experiments are too large to guide an ultrasound probe user to a precise location, the results indicate that neural networks with regression might be used for such purpose. To reduce the errors sufficiently, a much larger training set is probably needed. An extension of the regression technique could be to first use a classification network on an image to determine the body part (arm, leg, etc.), and then use a regression network trained on that particular body part to guide the probe user more precisely.

As in the synthetic experiments, changing the scaling factor in the loss function changed the individual errors in the two dimensions. The experiments show that applying a scaling factor in the loss and test functions is a viable way to compensate for the normalized image label coordinates, and, although the effect was less than expected, the scaling factors can be used to change the priority given to the two dimensions during training.

Although neural networks can be used to classify the approximate body location of ultrasound images, it is uncertain how fine-grained such classification might be. In the classification experiment performed on the lower spine dataset, the classification regions were much smaller than in the regional anesthesia experiments – between one and three centimeters – and a much lower accuracy was achieved in this case. Since the classification regions were small and adjacent to each other, the images in the different classes were rather similar. This made the task harder, and the lower accuracy was expected. This shows that using neural networks for classification of image locations may be more suitable for some areas of the body than others, but the method shows potential and should be further investigated. Also, a possible use of neural network classification might be to provide an approximate body location, and then use some other technique to further guide the ultrasound probe user, like a regression network.

7 Future Work

To improve the results obtained in this thesis, the most important factor would probably be to increase the number of persons in the training set considerably. By training on images from many more individuals, the network will be given a better chance to learn which features are important and common over a large population, making the network more robust. Also, when tested on a new person, the probability that the network has trained on a person with similar anatomy will increase.

Due to time limitations, only a few architectures and hyperparameter settings were tested in this thesis, but they produced rather different accuracies. This suggests that further architecture and hyperparameter testing has the potential to improve the results. This could for example include wider and deeper architectures; unlocking all layers of pre-trained networks; and investigating recurrent neural networks, operating on sequences of images instead of single images.

This thesis studied how well the networks could extract location information from raw image input. In a future study, investigations on which image features the networks made use of to predict these locations could be made. It is not in itself a goal to only use raw images as input, and learning which image features contributed the most to the localization prediction could possibly lead to more complex systems with increased accuracy.

In this thesis, precise image locations were obtained by using a regression network with coordinates as output. An alternative way to produce precise image locations would be to use a classification network with many classes, where each class mapped to a small rectangular area of the body, e.g. $1\ cm^2$. This alternative method was not selected for investigation in this thesis, as some problems were anticipated. For example, images originating from the border point of four adjacent regions could easily be predicted to belong to any of those four regions, creating a large area of uncertainty. Further experiments could shed more light on this problem and on the method as a whole.

In the lower arm experiments, a new model for the coordinate system could possibly improve the results. Since the lower arm is thicker near the elbow than at the wrist, one square unit of the normalized coordinates does not correspond to a rectangle, but rather to something like a trapezoid. In this thesis, the average half-circumference size of $12.5\ cm$ was used as the

latitudinal range, but a future experiment could vary this range according to the longitudinal probe position.

Data augmentation was only applied in the 1D synthetic experiments of this thesis, and only with standard augmentation methods. Although it did not reduce the prediction error of those experiments, data augmentation should be tested on the lower arm dataset, with the aim of increasing the dataset size and the generalization of the networks. Also, in addition to the standard augmentation methods, such as shifting and rotation, one might consider using (or developing) methods aimed at the ultrasound domain. This could e.g. include noise generation, simulating the typical noise found in ultrasound images.

In this thesis, the coordinate output of the regression method could be used to provide a distance and a direction to an arbitrary target coordinate. More valuable would be a system that could provide a distance and a direction to a specific *feature*, which might be located at slightly different coordinates on each person. As described in section 5.2, this could be achieved by assigning the center coordinate label to the images of that particular feature, and interpolate the rest of the label coordinates.

Alternatively, images aimed directly at the feature of interest could be assigned a zero coordinate (the origin), while images recorded around that feature could be given a coordinate describing the distance to the origin, in both the longitudinal and latitudinal directions. To make the labelling precise, an ultrasound probe with automatic tracking would probably be necessary. In addition, multiple recordings could be performed with varying probe rotation and tilting, and varying pressure against the skin. Each of these properties could be encoded as separate entries in the image labels. These entries could be given the value zero for images with optimal settings of rotation, tilting, and/or pressure, and positive or negative numbers for sub-optimal settings. A network trained on this dataset could be able to provide the ultrasound user guidance to the optimal view of the feature of interest, both in terms of probe location, rotation, tilting and pressure. With this approach, a separate network would be necessary for each feature of interest, or for each medical application.

The overall approach of such a system could be to first place the probe on the skin, near the feature of interest. Second, the system could automatically determine which feature the user was interested in looking at, by using a separate classification network trained for this purpose. Finally, the system could automatically apply the appropriate network for the selected feature,

to guide the user to the optimal view, as described above. The task of the user would then be to move and orient the probe such that all properties (all outputs of the network) were close to zero.

8 Bibliography

Alvarenga, A. V., Pereira, W. C. A., Infantosi, A. F. C., & de Azevedo, C. M. (2006). *GA-Backpropagation Hybrid Training and Morphometric Parameters to Classify Breast Tumors on Ultrasound Images.* Paper presented at the 6th IFAC Symposium on Modeling and Control in Biomedical Systems.

Arlot, S., & Celisse, A. (2010). A survey of cross-validation procedures for model selection. *Statistics Surveys, 4*(0), 40-79. doi: 10.1214/09-ss054

Becker, S., & Plumbley, M. (1996). Unsupervised Neural Network Learning Procedures for Feature Extraction and Classification. *Applied Intelligence, 6*(3), 185-203.

Burgstaller, B., & Pillichshammer, F. (2009). The average distance between two points. *Bull. Austral. Math. Soc. 80*, 353-359.

Carvalho, R. (2013). Average Distance Between Random Points on a Line. Retrieved 27. march, 2017, from http://math.stackexchange.com/questions/195245/average-distance-between-random-points-on-a-line

Chan, V., & Perlas, A. (2010). Basics of Ultrasound Imaging *Atlas of Ultrasound-Guided Procedures in Interventional Pain Management* (pp. 13-19): Springer New York.

Chiarazzo, V., Caggiani, L., Marinelli, M., & Ottomanelli, M. (2014). A Neural Network Based Model for Real Estate Price Estimation Considering Environmental Quality of Property Location. *Transportation Research Procedia, 3*, 810-817. doi: 10.1016/j.trpro.2014.10.067

Fossan, Ø. (2016). Using Neural Networks to Determine Body Locations of Medical Ultrasound Images (D. o. C. Science, Trans.): Norwegian University of Science and Technology.

Glorot, X., Bordes, A., & Bengio, Y. (2011). *Deep Sparse Rectifier Neural Networks.* Paper presented at the Proceedings of the 14th International Conference on Artificial Intelligence and Statistics.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*: MIT Press.

Google. (2016). TensorFlow. Retrieved 12.12.2016, 2016, from https://www.tensorflow.org/

Gray, A. T. (2006). Ultrasound-guided Regional Anesthesia: Current State of the Art. *Anesthesiology, 104*, 368-373.

Gupta, P. K., Gupta, K., Dwivedi, A. N. D., & Jain, M. (2011). Potential role of ultrasound in anesthesia and intensive care. *Anesthesia, Essays and Researches, 5*, 11-19.

He, H., & Garcia, E. A. (2009). Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering, 21*(9), 1263 - 1284.

Hinton, G., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580v1*.

Holm, S. (2008). Medisinks ultralydavbildning. *Fra Fysikkens Verden, 1*.

Jarret, K., Kavukcuoglu, K., Ranzato, M. A., & LeCun, Y. (2009). *What is the best multi-stage architecture for object recognition?* Paper presented at the IEEE 12th International Conference on Computer Vision.

Jones, D. (2016). PyPNG. Retrieved 29.05, 2017, from https://github.com/drj11/pypng

Keras. (2017). Keras: Deep Learning library for Theano and Tensorflow. from https://keras.io/

Kerby, B., Rohling, R., Nair, V., & Abolmaesumi, P. (2008). *Automatic Identification of Lumbar Level with Ultrasound*. Paper presented at the 30th Annual International IEEE Engineering in Medicine and Biology Society Conference.

Kingma, D. P., & Ba, J. L. (2015). *Adam: A Method for Stochastic Optimization*. Paper presented at the 3rd International Conference for Learning Representations.

Krizhevsky, A., Ilya, S., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25*.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436-444. doi: 10.1038/nature14539

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., W., H., & Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation, 1*, 514-551.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE, 86*, 2278-2324.

LeCun, Y., Cortes, C., & Burges, C. J. C. (2016). The MNIST database of handwritten digits. 2016, from http://yann.lecun.com/exdb/mnist/

Leshno, M., Lin, V. Y., Pinkus, A., & Schocken, S. (1993). Multilayer Feedforward Networks With a Nonpolynomial Activation Function Can Approximate Any Function. *Neural Networks, 6*(6), 861-867.

Liefers, B., Venhuizen, F. G., Theelen, T., Hoyng, C., van Ginneken, B., & Sánchez, C. I. (2017). *Fovea detection in optical coherence tomography using convolutional neural networks.* Paper presented at the Medical Imaging 2017: Image Processing.

LLC, V. (2014). Veebot LLC. from http://www.veebot.com

Lo, J. Y., & Floyd Jr, C. E. (1999). Application of Artificial Neural Networks for Diagnosis of Breast Cancer. *Proceedings of the 1999 Congress On Evolutionary Computation*.

Lundh, F., & Clark, A. (2016). Pillow - The friendly PIL fork. Retrieved 12.12.2016, 2016, from https://python-pillow.org/

Manbachi, A., & Cobbold, R. S. C. (2011). Development and application of piezoelectric materials for ultrasound generation and detection. *Ultrasound*, *19*(4), 187-196. doi: 10.1258/ult.2011.011027

Nair, V., & Hinton, G. E. (2010). *Rectified Linear Units Improve Restricted Boltzmann Machines*. Paper presented at the Proceedings of the 27th International Conference on Machine Learning, Haifa, Israel.

Nielsen, M. A. (2015). *Neural Networks and Deep Learning* Retrieved from http://neuralnetworksanddeeplearning.com/index.html

Noble, J. A. (2016). Reflections on ultrasound image analysis. *Medical Image Analysis, 33*, 33-37.

Paul, J. S., Plassard, A. J., Landman, B. A., & Fabbri, D. (2017). Deep learning for brain tumor classification. *Medical Imaging 2017: Biomedical Applications in Molecular, Structural, and Functional Imaging, 1013710*. doi: 10.1117/12.2254195

Perry, T. S. (2013). Profile: Veebot - Making a robot that can draw blood faster and more safely than a human can. from http://spectrum.ieee.org/robotics/medical-robots/profile-veebot

Simonyan, K., & Zisserman, A. (2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. Paper presented at the ICLR.

Singh, B. K., Verma, K., & Thoke, A. S. (2016). Fuzzy cluster based neural network classifier for classifying breast tumors in ultrasound images. *Expert Systems With Applications, 66*, 114-123.

Smistad, E., & Løvstakken, L. (2016). *Vessel detection in ultrasound images using deep convolutional neural networks.* Paper presented at the Deep Learning and Data Labeling for Medical Applications.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from

Overfitting. *Journal of Machine Learning Research, 15*.

Stanford_Vision_Lab. (2016). ImageNet. from http://image-net.org/

Stern, B. (2016). The Basic Concepts of Diagnostic Ultrasound1. Retrieved 12.12.2016, 2016, from http://teachersinstitute.yale.edu/curriculum/units/1983/7/83.07.05.x.html

Taylor, P. M. (2003). Ultrasound for anaesthetists. *Current Anaesthesia & Critical Care, 14*(5-6), 237-249. doi: 10.1016/j.cacc.2003.11.001

The MathWorks, I. (2017). Train a Convolutional Neural Network for Regression. Retrieved 07.04, 2017, from https://se.mathworks.com/help/nnet/examples/train-a-convolutional-neural-network-for-regression.html

TorontoWesternHospital. (2017). Regional Anesthesia > Specific Blocks. Retrieved 03.05.2017, 2017, from http://www.usra.ca/regional-anesthesia/specific-blocks/home.php

Xue, Y., Ray, N., Hugh, J., & Bigras, G. (2016). *Cell Counting by Regression Using Convolutional Neural Network*. Paper presented at the Computer Vision – ECCV 2016 Workshops.

Yale_University. (2016). Computer vision: LeNet-5, AlexNet, VGG-19, GoogLeNet. Retrieved 28.02, 2017, from http://euler.stat.yale.edu/~tba3/stat665/lectures/lec18/notebook18.html

Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems 25, 27*, 3320-3328.

Yu, S., & Tan, K. K. (2014). *Classification of Lumbar Ultrasound Images with Machine Learning*. Paper presented at the Simulated Evolution and Learning.

9 Appendices

9.1 Description of Keras Layers

Here follows a brief explanation of the Keras code used to describe neural networks in this thesis.

```
Convolution2D(x, y, z)
```

This layer applies a 2-dimensional convolution. `x` specifies the number of convolutions kernels to apply simultaneously, while `y` and `z` specifies the kernels' width and height, respectively.

```
MaxPooling2D(pool_size=(2, 2), strides=(2,2))
```

This layer applies max pooling. `pool_size` specifies the width and the height of the window in which to perform max pooling, while `strides` specifies the stride step to take between each max pooling window.

```
Activation('relu')
```

This layer applies the Relu activation function.

```
Activation('softmax')
```

This layer applies the softmax activation function.

```
Dense(x)
```

This applies a fully connected (dense) layer, where `x` is the number of nodes in the layer.

```
ZeroPadding2D((x,y))
```

This layer pads the output of the previous layer with zeros. `x` specifies the number of zeros to pad to the top and bottom, while `y` specifies the left and right pad.

```
Flatten()
```

This is a function that flattens the 2-dimensional node matrix from the previous layer into a 1-dimensional vector of nodes.

```
Dropout(x)
```

This layer applies the dropout regularization technique to the output of the previous layer, by randomly dropping a specified fraction of the nodes. The fraction is given by `x`.

```
input_shape = (x,y,z)
```

This parameter is always added to the first layer of the network. It states the shape of the network input, such that `x` and `y` is the height and width, while `z` is the number of channels.

9.2 Keras Implementation of AlexNet

```
# Layer 1
Convolution2D(96, 11, 11, input_shape = (28,28,1))
Activation('relu')
MaxPooling2D(pool_size=(2, 2))

# Layer 2
Convolution2D(256, 5, 5)
Activation('relu')
MaxPooling2D(pool_size=(2, 2))

# Layer 3
ZeroPadding2D((1,1))
Convolution2D(512, 3, 3)
Activation('relu')

# Layer 4
ZeroPadding2D((1,1))
Convolution2D(1024, 3, 3)
Activation('relu')

# Layer 5
ZeroPadding2D((1,1))
Convolution2D(1024, 3, 3)
Activation('relu')
MaxPooling2D(pool_size=(2, 2))

# Layer 6
Flatten()
Dense(3072)
Activation('relu')
Dropout(0.5)

# Layer 7
Dense(4096)
Activation('relu')
Dropout(0.5)

# Layer 8
Dense(10)
Activation('softmax')
```

9.3 Keras Implementation of VGG16

```
# Layer 1
ZeroPadding2D((1,1) input_shape = (224,224,3))
Convolution2D(64, 3, 3)
Activation('relu')
ZeroPadding2D((1,1)
Convolution2D(64, 3, 3)
Activation('relu')
MaxPooling2D(pool_size=(2, 2), strides=(2,2))

# Layer 2
ZeroPadding2D((1,1))
Convolution2D(128, 3, 3)
Activation('relu')
ZeroPadding2D((1,1)
Convolution2D(128, 3, 3)
Activation('relu')
MaxPooling2D(pool_size=(2, 2), strides=(2,2))

# Layer 3
ZeroPadding2D((1,1))
Convolution2D(256, 3, 3)
Activation('relu')
ZeroPadding2D((1,1))
Convolution2D(256, 3, 3)
Activation('relu')
ZeroPadding2D((1,1))
Convolution2D(256, 3, 3)
Activation('relu')
MaxPooling2D(pool_size=(2, 2), strides=(2,2))

# Layer 4
ZeroPadding2D((1,1))
Convolution2D(512, 3, 3)
Activation('relu')
ZeroPadding2D((1,1))
Convolution2D(512, 3, 3)
Activation('relu')
ZeroPadding2D((1,1))
Convolution2D(512, 3, 3)
Activation('relu')
MaxPooling2D(pool_size=(2, 2), strides=(2,2))

# Layer 5
ZeroPadding2D((1,1))
Convolution2D(512, 3, 3)
Activation('relu')
ZeroPadding2D((1,1))
Convolution2D(512, 3, 3)
Activation('relu')
ZeroPadding2D((1,1))
Convolution2D(512, 3, 3)
Activation('relu')
MaxPooling2D(pool_size=(2, 2), strides=(2,2))

# Layer 6
Flatten()
Dense(4096)
Activation('relu')
```

```
Dropout(0.5)

# Layer 7
Dense(4096)
Activation('relu')
Dropout(0.5)

# Layer 8
Dense(1000)
Activation('softmax')
```