Deep Learning with a Directed Acyclic Graph Structure for Segmentation and Classification of Prostate Cancer

Gabrielle Flood

Master's thesis
2016:E44

LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

LUND UNIVERSITY

Master's Thesis, 2016
Mathematics

Deep Learning with a Directed Acyclic Graph Structure for Segmentation and Classification of Prostate Cancer

Gabrielle Flood

Supervisor: Kalle Åström

Lund University
Faculty of Engineering
Centre of Mathematical Sciences

Abstract

Deep learning is a machine learning technique inspired by the biological nervous system. The method has been used more and more over the last decades. Within this thesis, convolutional neural networks (CNN:s) are used for Gleason classification of prostate cancer in histopathological images. The data that has been used consists of digitalized microscopic images of prostate biopsies stained with haematoxylin-eosin (H&E). There are separate samples containing benign, Gleason 3, Gleason 4 and Gleason 5 tissue. More specifically, the potential of a directed acyclic graph (DAG) structure for the classifying CNN has been investigated. In this network a segmentation image has been used as a second label in the middle of the network, and the backpropagation has thus been performed using two different outputs. Hence, as a buildup, a CNN with a simple structure was created for segmentation of different components in benign prostate tissue. The structure and trained weights of this segmentation network were then used in the classifying DAG net. No final fine-tuning of the hyper-parameters or optimization of the network has been performed, but the cost function showed a decreasing trend when the training was carried out. Therefore it was concluded that the DAG structure does have potential and that further development of the method could yield a high performance network.

Acknowledgments

I would like to thank all the people who have helped during the work with my master's thesis. First , I would like to express my sincere gratitude to my supervisor Kalle Åström. Thank you for giving me a lot of help, but not too much. Thank you for introducing me to the field of deep learning and for acting like a sounding board when I myself have been confused. My work, my learning and my development would not have been the same without you.

I would also like to thank Agnieszka Krzyzanowska and her colleagues for inviting to a visit at the pathology department at Skåne University Hospital. The medical information gave me understanding and also a great motivation.

Finally I wish to express my gratitude to the other thesis workers at the Department of Mathematics. Thank you for making the coffee breaks a great time for revitalization, but also for listening and helping when problems with my thesis have arisen. If I have colleagues like you in the future, I will be happy. A special thanks to Anna Gummeson who have been working on a similar problem as I have. Thank you for answering my sometimes very basic questions and discussing both mine and your problems. You have given me a better insight into the field of neural networks.

Contents

1 Introduction

The human brain consists of 100 billions of neurons, with up of up to 10000 synapses each [1]. A human can without difficulties learn to recognize things such as faces, environments, numbers and patterns. The human brain is also very good at drawing logical conclusions, where one thing leads to another based on certain fact. Even though these are obvious abilities for a human brain, the tasks can be remarkably complex to program. The main idea behind artificial intelligence is to make a computer think, act, draw conclusions and make decisions in the same way as humans do. The vision is that if a computer program could be designed to use as many neurons as a human brain, it could also solve as complicated problems as humans can. Today, there is not enough computational power to do that, but in the future there might be and the method can still be very powerful.

Just like biological neural networks, the artificial neural networks are built up by neurons, which in the artificial case also are called *nodes*. These are usually ordered and connected to each other in certain ways. The connections are also inspired by the ones in biological networks and these can e.g. make the nodes be on or off, or transfer a passing signal less or more [2].

Convolutional neural networks (CNN:s) is one type of artificial neural networks that is commonly used to perform classification of images. The data has another structure than it does for regular neural networks and the linear operations performed within the network are mainly built on 2D convolution. Furthermore, the operations — which can be non-linear as well — are performed in several steps or *layers*. Due to this, the method is also referred to as *deep learning*. The theory behind deep learning will be explained in Chapter 3.

This thesis has as a purpose to investigate potential structures for deep neural networks. The networks should be used on microscopic images of prostate biopsies to classify whether the samples contain cancer. If they do, the samples should be graded according to the Gleason grading system for prostate cancer [3]. A brief overview of prostate cancer and the used grading systems is given in Chapter 2.

In the present situation the Gleason classification of prostate biopsies is performed manually by pathologists. First of all, this is a time consuming work. Secondly there is a shortage of pathologists in Sweden at the moment [4]. Due to this, potentiation of the screening would be of great importance. An automatic Gleason classification method could as a first step be used as a recommendation to the pathologists. After running the images through the network the physician could be told which samples to take a closer look at. In the future, maybe the computer could handle the whole classification. The automatic classifier could also be used as a second opinion, since the Gleason grading scale is not very strictly defined and the opinions of different pathologists usually differ somewhat.

1.1 The DOGS Project

DOGS — Digital Pathology for Optimized Gleason Score in Prostate Cancer — is a project project financed by Vinnova [5]. The aim of the project is to develop computerized and automatic evaluation of Gleason grading. The project runs during the years 2015 to 2017 and is carried out by the Department for Translational Medicine at Lund Universiy. This thesis is a small part of that project and could be seen as a pre-study to future work within the DOGS project.

1.2 Related Work

The idea of an automatic system for classification of prostate tissue is not new. There are several examples where this has been investigated in different manners. A few of them will be mentioned here.

Lippolis et al. tried to classify prostate tissue on a Gleason grade from 1 to 5. This was done using scale-invariant feature transform, bag of words and support vector machine. A binary classification was performed, where the algorithm should distinguish between two different grades or groups or grades. E.g. an accuracy of 98.1% was achieved when trying to distinguish between benign and cancerous tissue and in a one-vs-all multiclass scheme, 87.3% accuracy was achieved. [6].

Källén et al. used pre-trained deep neural networks and random forest or support vector machine to classify prostate tissue as benign, Gleason 3, Gleason 4 or Gleason 5. The deep convolutional network that was used there was pre-trained on a large set of photographic images and then adapted to microscopic prostate images. The network was then terminated at an earlier layer, whereupon the achieved output was used to train both a random forest classifier and a support vector machines classifier. Källén et al. achieved an accuracy of 89% for the prostate images that were tested. [7]

Litjens et al. tried to detect both prostate cancer in biopsies and breast cancer metastasis in resected sentinel lymph nodes. That network was trained to perform binary classification to distinguish between benign and cancer. The network was trained from scratch using cut out patches from the original images. This article focuses both on efficiency and accuracy and the results for the prostate cancer detection is presented as a ROC-curve measuring sensitivity and specificity. [8]

Gummeson had a similar approach as Litjens, but was looking only at microscopic prostate biopsies. That network was also trained from scratch but to distinguish between benign, Gleason 3, Gleason 4 and Gleason 5 tissue. Gummeson achieved an accuracy of 92.7%. [9]

1.3 Aim of and Goals with the Thesis

The main goal of this thesis has been to investigate the potential of different CNN structures that could be used for an automatic Gleason classification system. More specifically, these structure were built using a directed acyclic graph (DAG) CNN. At the same time another important aim has been to develop the understanding for deep neural networks and different CNN structures — simple and directed acyclic graph (see Chapter 3.3).

The final network was supposed to be built as a straight network, with one outgoing extra branch with a connected *middle key* and this middle key is also the main difference from the work presented by Litjens [8] and Gummeson [9]. The final classification key, or *label* is thus made up of the Gleason grade, while the middle key is a segmentation image of different tissue parts. This is a method which has not been used on the images in the project before and the thesis was therefore supposed to investigate whether this would be a promising approach. The project has been divided in two subprojects, one segmentation problem and one classification problem.

1.3.1 The Segmentation Problem

The first subproject, the segmentation problem, was performed using a simple neural network. The point of this part was to develop an initial understanding for deep neural networks and to see whether the middle key could be useful in the final, classifying network. The segmentation problem only uses benign images and was supposed to perform segmentation of certain parts of the prostate tissue in the images.

1.3.2 The Classification Problem

The second subproject, the classification problem, had as an aim to fulfill the main goal of the thesis. Thus, a directed acyclic graph CNN was used. The idea was that one part of the network should be inspired by the simple network used for the segmentation problem, so that the middle key could be a segmentation image. This segmentation should be a “branch” from the rest of the network, which finally led to the Gleason grading, or tissue classification. The potential of the architecture was to be analyzed and the short-term goal was thus not to achieve a perfect classifier.

2 Prostate Cancer

The prostate is an organ only present in male bodies. It usually has the size of a walnut and is located below the bladder with the urethra going through it [6]. Prostate cancer is the most common cancer for men and about 35% of all cases of cancer for Swedish men are prostate cancer. For a Swedish man, the risk of getting prostate cancer is approximately 18% and around 2500 men in Sweden die every year as an effect of prostate cancer [3]. Prostate cancer mainly affect elderly men — the average age at detection is around 70 years old — but it is quite common for men over 50. It has also become more common that the cancer is found in an earlier stage.

2.1 Prostate Tissue

Within this report some parts of the prostate tissue will be mentioned and used. An example of a part of a prostate biopsy can be seen in Figure 2.1.

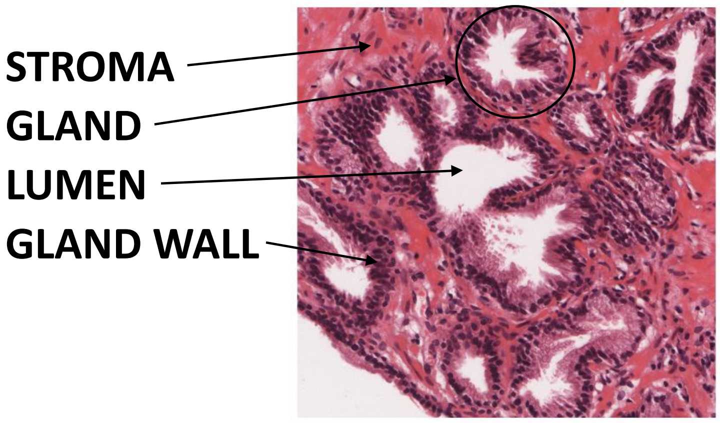

Figure 2.1: An example of normal prostate tissue, with some parts of the tissue marked.

Normal prostate tissue consists of *glands* and *stroma*. The glands are made up of an empty lumen, surrounded by cells which are creating a *gland wall*. The lumens appear as white or pale on the microscopic images used within this thesis, as the ones in Figure 2.1. There are also certain structures in the gland wall, see further [6]. The stroma is the rest of the prostate tissue, what surrounds the glandular structures. This is the part that is pink in the image above.

2.2 Cancer Tests and Diagnostics

In general, prostate cancer can develop quite far without showing any symptoms. The first test that is done is a blood test which looks for increased levels of prostate specific antigen, PSA. The PSA is a protein which is produced in the prostate when it is healthy as well, but when cancer occurs PSA leaks into the blood in a greater occurrence. Through a blood sample the prostate cancer can thus be detected in an early stage, before any symptoms occur. Though, there are also cases of benign enlargements of the prostate and these can also result in an increased PSA level. That means that a positive blood sample not necessarily denotes cancer.

If a blood sample shows increased levels of PSA the investigation is taken one step further. In some cases, the tumor is large enough for the doctor to feel it through an rectal examination. Otherwise, transrectal ultrasound can be used. In that case, biopsies are usually taken as well. The biopsies are taken using needles and ten or twelve samples, each a few centimeters long, are taken from different parts of the prostate. After that, in some cases when the cancer is bad, the whole prostate is removed. That is called radical prostatectomy. Thence the prostatectomy is not performed for screening but as a medical intervention.

Independently of whether the samples come from needle biopsy or radical prostatectomy, the biopsies are afterwards studied in microscope by a pathologist. This is done to determine whether cancer is present (in the case of needle biopsy) and in that case how far the disease has gone. When cancer develops the glandular structure in the prostate is decreased or lost and depending on how much of the structure that is gone, how aggressive the cancer is and how likely it is to spread, the cancer is assigned a certain grade.

2.3 The Gleason Grading System

The system used to classify cancer is called the Gleason grading system or the Gleason classification system. The system was developed around the 1970's by Donald Gleason. The original scale goes between 1-5, with Gleason grade 5 describing the most aggressive cancer [10]. An explanation of the Gleason grading scale is given in Figure 2.2. Though, later on pathologists have realized that Gleason grade 1 does not exist, even if it resembles benign tissue. Furthermore, Gleason grade 2 does exist, but can only be classified if a large area of the prostate is analyzed, like when the whole prostate is removed. Thus, the Gleason grade 2 will not show on the needle biopsies and therefore that grade is never used in reality since the radical prostatectomy is not performed right away or if the cancer is as "kind" as Gleason 2.

The first grade that is actually used is therefore Gleason grade 3. In Gleason grade 3 tissue, there are still clear glandular structures, but the glands are generally smaller than for benign tissue. Furthermore, the size and shape of the glands can vary a lot.

For Gleason grade 4 the glandular structure has disappeared even more and it is hard to distinguish the single glands. Instead the glands have started to coalesce. Except

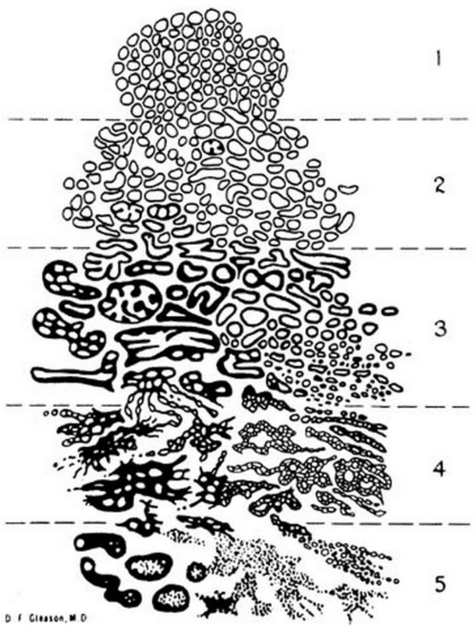

Figure 2.2: A description of the Gleason grading system. The worse the cancer is, the more are the glandular structures lost. Gleason grade 5 indicates the worse case of prostate cancer.

for that they are even smaller than for Gleason 3 and the lumens have substantially narrowed off.

When it comes to Gleason grade 5 the structure is almost completely gone and the tissue does not resemble healthy prostate tissue at all. For benign prostate tissue, only stromal cells are lying alone, while glandular cells are staying together. For Gleason 5 there are a lot of glandular cells that are separate from others and there is no differentiation between the glands at all. Thus, the glands cannot be distinguished and the lumens are completely gone.

Figure 2.1 shows an example of benign tissue and Figure 2.2 shows the different grades of cancer. In this project, the classes mentioned above — that is benign and Gleason 3, 4 and 5 — have been assigned.

2.3.1 The Gleason Score

When pathologists speak about prostate cancer they usually do not refer to the Gleason grade, but the Gleason score. The Gleason score is achieved by adding the two most common grades in a tumor, so in theory it ranges from 2-10, but in reality only from 6-10. For example, if a patient with prostate cancer mainly has Gleason 4 in the tumor, but the second most common grade is Gleason 3, then the final Gleason score will be 4 + 3 = 7.

2.3.2 The Gleason Grade Group

Another way to assign how bad the prostate cancer is using the Gleason grade is the Gleason group. This was introduced in the 2014 International Society of Urological Pathology (ISUP) Consensus Conference. One reason for the new system was the fact that the lowest assigned Gleason grade is a 6 out of 10 and that this can make the cancer sound worse than it actually is. Another motivation was that when the Gleason score is used, there is no way of distinguishing between Gleason $3 + 4$ and Gleason $4 + 3$, since both would be Gleason score 7. In reality though, the latter would mean a worse state than the former. The Gleason grade groups reaches from 1 to 5 and the division is clarified in Table 2.1. [11]

Table 2.1: The Gleason grade group system. The different groups defined using the Gleason grade and score. The values are taken from [11].

Grade group 1	Gleason score ≤ 6
Grade group 2	Gleason score $3 + 4 = 7$
Grade group 3	Gleason score $4 + 3 = 7$
Grade group 4	Gleason score 8
Grade group 5	Gleason score $9 - 10$

3 Artificial Intelligence and Neural Networks

An artificial neural network is — just like the neural network in the human brain — built up of neurons. In the artificial case the neurons are also called nodes. The nodes can be divided in different stages or layers. The nodes of one layer can “communicate” with the nodes in the previous and the following layer in the same manner as biological neurons do. Biological neurons are further explained in [2].

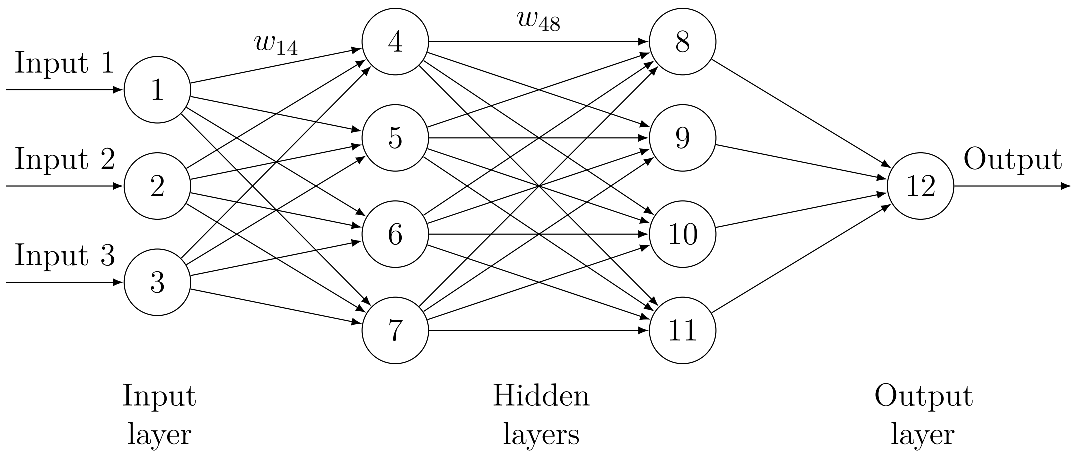

Figure 3.1: A neural network with four layers; one input layer, two hidden layers and one output layer. The nodes are marked by circles and the connections between the nodes by arrows. Above two of the connections, the weights w are shown.

A network contains three different kinds of layers; an input layer, an output layer and hidden layers. The hidden layers are all layers that are not the input or output layer. A small example of a network can be seen in Figure 3.1. The network has four layers, of which two are hidden, and in total the network contains twelve nodes.

Now the focus will be on one of the nodes in the network, node number 4. Regarding node number 4 there are a two functions of interest; the propagation function and the activation function [1]. The propagation function concerns the inputs, in this case coming from node 1, 2 and 3. Often the propagation function is a weighted sum of the inputs, such that

$$p_j = \sum_i w_{ij} a_i, \tag{3.1}$$

where w_{ij} is the weight from node i to node j and a_i is input number i (that is, input to the current node, not to the network). In the case of node number 4, the propagation function would look like

$$p_4 = \sum_{i=1,2,3} w_{i4} a_i. \tag{3.2}$$

It is also common that a bias term is added to the weighted sum, which gives the final propagation function

$$p_j = b_j + \sum_i w_{ij} a_i. \tag{3.3}$$

The propagation function between all nodes have the same structure, but the values of the weights and biases differ.

The activation function takes care of the output of the propagation function and decides whether a node is “on” or “off”, that is compared to a physical neuron whether it will fire [2]. A simple activation function could be a threshold function, where

$$a_j = \begin{cases} 0 & \text{if } p_j < \text{threshold}, \\ 1 & \text{if } p_j \geq \text{threshold}. \end{cases} \tag{3.4}$$

In the case where the bias term is added to the propagation function, Equation (3.3), the threshold could simply be set to 0, since there is no need for having two constants. The bias is enough to control the activation threshold.

A disadvantage with a simple step function, which is given by the threshold in Equation (3.4), is that the output changes from 0 to 1 very quickly. Furthermore, the step function cannot be continuously differentiated. To overcome this problem, other activation functions are often used. One is the sigmoid function, defined as

$$f(x) = \frac{1}{1 + \mathrm{e}^{-x}}. \tag{3.5}$$

Another common activation function is the hyperbolic tangent function

$$f(x) = \frac{\mathrm{e}^{x} - \mathrm{e}^{-x}}{\mathrm{e}^{x} + \mathrm{e}^{-x}}. \tag{3.6}$$

The sigmoid function and the hyperbolic tangent function have very similar shape and are re-scaled versions of each other, the sigmoid having an output range between 0 and 1 and the tanh having an output range between -1 and 1, see Figure 3.2.

There is also another activation function that is common, namely the rectified linear function [12]. Compared to the sigmoid and the hyperbolic tangent, the rectified linear function has no upper bound and it is not continuously differentiable. The function is given by

$$f(x) = \max(0, x). \tag{3.7}$$

A plot of the different activation functions can be seen in Figure 3.2. Usually, the activation function is global and thus the same in all parts of the network [13]. If one of the later activation functions is used instead of the threshold the activation will be $a_j = f(p_j)$. In this thesis the rectified linear function has been used.

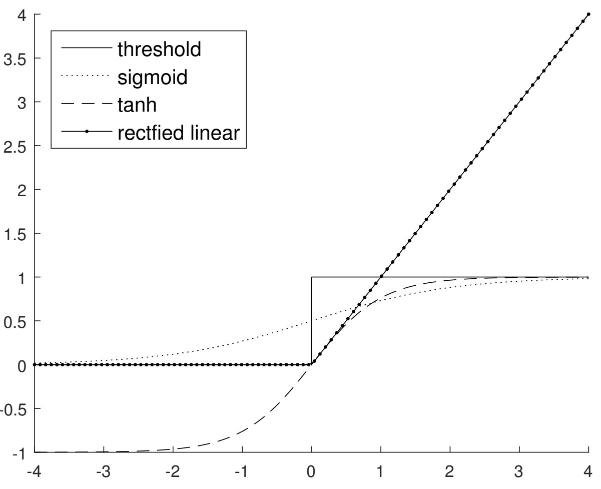

Figure 3.2: Four different activation functions, one step function created by simple threshold (Equation (3.4)), the sigmoid function (Equation (3.5)), the hyperbolic tangent function (Equation (3.6)) and the rectified linear function (Equation (3.7)).

It might seem a bit confusing that the node inputs in Equation (3.3) are denoted with a:s, just as the activations are, e.g. in Equation (3.4). The reason for this is of course that the activations of one layer are the inputs to the next layer and thus the notation that at first sight might seem ambiguous actually makes sense.

The signal passes through all nodes in a similar way as described above — with the activation of one layer passed to the next, to yield first the propagation and then the activation of that layer and so forth — until the total network output is retrieved. This type of network, which lets a signal pass through a whole network, one layer at a time, is called a feedforward neural network [12]. One thing that could be worth noting is that a network can have any number of nodes in the input and output layer, not necessarily three inputs and one output, as in Figure 3.1.

3.1 Training the Network

3.1.1 Supervised or Unsupervised Training

Once a feedforward network is initialized, it can be trained to do correct predictions. The training can be either unsupervised or supervised. In the unsupervised case the network should be provided input training data $(x_1, x_2, ..., x_n)$. These inputs x_i can be of different dimensions. In the network in Figure 3.1, $x_i \in \mathbb{R}^3$, since node 1, 2 and 3 are parts of the input. Thus, in general $x_i \in \mathbb{R}^m$ where m is the number of nodes in the input layer. Though, there will later be cases where the input nodes are arranged as an image.

After the network has been provided the input it will cluster the training data in a given number of groups and hopingly catch interesting structures. After running the network on enough training data, it will be good at the clustering and then, if a new input is added for testing, it will hopefully be correctly classified.

In the case of supervised learning the network should be provided pairs of input data and desired output, $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$. The desired output y_i constitutes the classification ground truth for the training data, that is the label that x_i should be assigned to be correctly classified. The label y_i is also referred to as the *key*. The inputs x_i have the same dimensions as in the unsupervised case, while $y_i \in \mathbb{R}^\gamma$, with γ being the number of possible classes. The label y_i contains a 1 at the position corresponding to the correct class, and 0:s at all other positions. For example, $y_i = [0\,1\,0\,0]^T$ indicates that x_i belongs to class number two.

The training is then performed by running the training data through the network and comparing the output with the provided ground truth, to see how good the network is. The network is then adjusted to make the error decrease — which there will be more about in the next section. As in the case of unsupervised training, the test data does not need any ground truth.

There is also a third kind of training which is a combination of supervised and unsupervised training, where some of the input data has a ground truth. The method is called semi-supervised training. More information about that can be found in [14].

In this project, only the supervised learning has been applied since the used data has provided labels.When training is discussed further on this will thus refer to the supervised training.

3.1.2 The Cost Function

As mentioned above, the supervised training is such that it learns by comparing the network output to the ground truth and then adjusting the weights and biases in the network according to that. Thus, the first thing that is needed is a function that takes care of the comparison, a loss function $l(y, \hat{y})$ which expresses the penalty of predicting $\hat{y}$ instead of desired output y [15]. One simple example of such a loss function is

$$l(y, \hat{y}) = ||y(x_i) - \hat{y}(x_i)|| = \sum_{k=1}^{\gamma} ||y_k(x_i) - a_k(x_i)||, \tag{3.8}$$

where y is the key and y_k the component of that for output neuron k. Furthermore $\hat{y}$ is, as mentioned, the predicted output, which is made up of the activations a_k of the output layer — so the activations from neurons k. The x_i symbolizes that it is the output given by running input x_i through the network.

Furthermore, using this loss function a function that takes care of the comparison and penalization for the whole network can be composed. This function will be called the cost function $C(w, b)$ — but is also known as the error function or the objective function. The terms will be used interchangeably. The cost function depends on the weights in the network, w, and the biases, b. Of course, the function will also contain the true and predicted outputs, but these are considered constant, since it is the weights and biases that are supposed to be adjusted. The cost function looks as follows

$$C(w,b) = \frac{1}{n}\sum_{i=1}^{n} l(y(x_i), \hat{y}(x_i)). \tag{3.9}$$

Here, n is the number of training inputs. Since the loss function contains all output neurons, the cost function is a summation over all outputs of all inputs. It is thus adjusted to work for different kinds of networks. The factor in front of the sum can differ but for the minimization that will not matter.

It is also common to add a term to the cost function to penalize large weights and this can be done independently of how the original cost function looks. This is called weight decay, or L2 regularization [16]. The regularization term consists of the sum of the squared weights, scaled by a regularization parameter λ. With that term, the regularized cost function $\tilde{C}$ would look like

$$\tilde{C}(w,b) = \frac{\lambda}{2n}\sum w^2 + C(w,b), \tag{3.10}$$

where the summation is over all weights in the network. The exact loss function that has been used within this thesis will be presented later.

3.1.3 Gradient Descent Method

Once the cost is computed the actual training step can be performed. This is done using the so called backpropagation method, which is based on the gradient descent method. When a function has many variables, explicit calculations of the global minimum can be troublesome. That is why the gradient descent method is used in the case of neural networks. The cost function $\tilde{C}$ could be seen as a landscape, where the aim is to find the deepest valley. To do this, several small steps are taken. Every step is taken such that it goes in a down slope direction. Sooner or later, the valley will be reached.

In mathematical terms, this is approached by taking steps in the direction of the negative gradient. Once the value of the function to be minimized is computed, the gradient $\nabla\tilde{C}(w,b)$ in the current point (w,b) is computed. The gradient tells in which direction the function changes the most and therefore, a step is taken in the negative gradient direction. This step is taken through a change of the variables, so all the weights and biases of all layers in the network are adjusted. The size of the step taken is controlled by a learning rate variable η. The variables are updated according to [15]

$$w_{t+1} = w_t - \eta\frac{\partial\tilde{C}}{\partial w} = w_t - \eta\frac{\partial C}{\partial w} - \eta\frac{\lambda}{n}, \tag{3.11}$$

$$b_{t+1} = b_t - \eta\frac{\partial\tilde{C}}{\partial b} = b_t - \eta\frac{\partial C}{\partial b}. \tag{3.12}$$

Equations (3.11) and (3.12) show the simplest version of the gradient descent algorithm. Though, there are more things that can be good to take into account, in addition to the direction of the steepest descent. Therefore, there will be an additional term to Equations (3.11) and (3.12), called a momentum term [15]. The momentum works as a "memory", and by using it the update of the parameters will be a combination of the current gradient and the previous update. The momentum term is present to reduce oscillations around the minimum if the valley is particularly steep and narrow. With the momentum term, the update at each time step is done in two steps: first by updating the "changing terms" Δw and Δb — which earlier was given by the scaled gradient — and then updating the parameters. The momentum variable μ controls how much memory the network should have. The update of the weights will then work as follows [15]

$$w_{t+1} = w_t - \Delta w_{t+1}, \qquad \Delta w_{t+1} = \mu \Delta w_t + \eta \frac{\partial \tilde{C}}{\partial w_t}. \tag{3.13}$$

The update of the biases will be done in a similar matter. Note that the cost function $\tilde{C}$ refers to Equation (3.10), which includes the regularization term.

Stochastic Gradient Descent Method

There is a version of the gradient descent method which is called stochastic gradient descent [16]. In the regular method which is discussed above the update is done based on the cost and the gradient over the whole training set. In the stochastic version only a single, or a minibatch of size τ of the training inputs is used to compute the gradient of the parameters. This will result in an approximation of the actual gradient, but is much faster to compute. By taking a new, random batch out of the rest of the inputs in each time step the whole training set is used. All inputs have to be used a first time, before any is used a second time. When all inputs have been included in the minibatch one time, an *epoch* is said to have passed. Due to this, the parameters are still adapted to all different examples but the time requirement is reduced.

3.1.4 The Backpropagation Algorithm

Computing the updates of the parameters might seem simple at first glance, but in fact, the partial derivatives $\partial\tilde{C}/\partial w$ and $\partial\tilde{C}/\partial b$ take a lot of effort to compute. Moreover, this is supposed to be done for each weight and bias in the network, which can be a great many in a large network. The backpropagation algorithm diminishes the computational effort needed for these calculation.

Before explaining the actual algorithm some new indexations will be introduced. Recall the network from Figure 3.1. Instead of just numbering all the nodes from the first to the last, first the layers will be numbered and then the nodes in each layer. Each of the layers 1 to 4 will for example contain a node number 1, see Figure 3.3. The different nodes numbered with a "1" are thus not the same. Furthermore, the weight w^l_{ij} goes to node number j in layer l from node number i in layer $l-1$. In the same manner, the bias added to node number j in layer l is denoted b^l_j and the activation of that node is denoted a^l_j. With these new notations, a certain activation will be computed as

$$a^l_j = f\left(b^l_j + \sum_i w^l_{ij} a^{l-1}_i\right), \tag{3.14}$$

where $f(x)$ can be one of the mentioned activation functions, either the sigmoid in Equation (3.5), the hyperbolic tangent in Equation (3.6) or the rectified linear function in Equation (3.7). The sum in the activation equation above is over all the nodes i in layer $l - 1$.

Using these new notations, the weights, biases and activations can easily be arranged in matrices. For each layer l there will be a weight matrix w^l whose entry on row i, column j will be w^l_{ij}. That means that column j in the weight matrix w^l will contain all the weights used in the computation of the activation for node number j in layer l. Furthermore, b^l and a^l will be vectors for the biases and activations of layer l and the number at place i in the vectors correspond to node number i in the layer. By now, Equation (3.14) can be rewritten in matrix form such that the activations for a whole layer are computed at once

$$a^l = f\left(b^l + (w^l)^T a^{l-1}\right). \tag{3.15}$$

The function $f(x)$ is the same as before, here working elementwise on the matrix it is applied to. Just in the same way as before the activation is achieved by applying the activation function to the propagation, see Equation (3.3). The propagations for the nodes in layer l can then also be expressed using matrix notations, such that

$$p^l = b^l + (w^l)^T a^{l-1}, \tag{3.16}$$

where p^l is the propagation matrix. Using this notation would thus give $a^l = f(p^l)$, just as before.

Now it is time to start with the actual backpropagation. First of all, an investigation will be made concerning what effect a small change at a certain node has. Say that the propagation of node j in layer l is changed by a small number Δp^l_j such that the output of that node is changed from $f(p^l_j)$ to $f(p^l_j + \Delta p^l_j)$. This small change would then propagate through the later layers of the network and result in a change of the cost function of the size $\partial C / \partial p^l_j \cdot \Delta p^l_j$. If this change Δp^l_j is inserted in the network it can be used to control the resulting cost. With this as a motivation, the intermediate quantity

$$\delta^l_j = \frac{\partial \tilde{C}}{\partial p^l_j} \tag{3.17}$$

is introduced [16]. The error δ^l_j is a measure of how much the j:th node in layer l contributes to the total cost at the end of the network [16]. In the same manner as before, δ^l is a vector containing all the errors of layer l.

The backpropagation algorithm can be built up from four essential equations. The full proofs will not be given here, but can be found in [16]. The equations will be explained briefly.

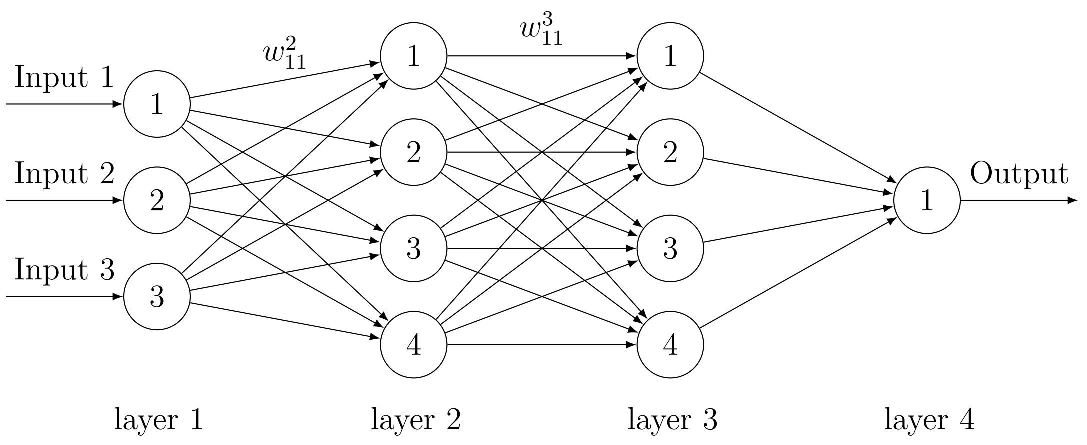

Figure 3.3: The same network as in Figure 3.1, but with a different indexation. The network contains four layers of which two are hidden. Node number 1 in layer 1 is not the same as node number 1 in layer 2 and so forth. Also, notice the change of weight notation.

The Error in the Output Layer

The error for a certain node in a certain layer is given from Equation (3.17). For the j:th node in the last layer in the network – layer L – the error will be

$$\delta_j^L = \frac{\partial \tilde{C}}{\partial p_j^L} = \frac{\partial \tilde{C}}{\partial a_j^L} f'(p_j^L), \tag{3.18}$$

where the last step is given by the the chain rule, since $a_j^L = f(p_j^L)$. The equation above is thus a measure of how much the cost function changes as a function of the propagation of neuron j in the last layer. What the derivatives will be depends on how the cost function looks and which activation function that is used.

As before, it is desirable to have the equations in matrix form. The matrix form of Equation (3.18) is

$$\delta^L = \nabla_a \tilde{C} \odot f'(p^L), \tag{3.19}$$

with $\nabla_a \tilde{C}$ being a vector whose components are given by $\partial \tilde{C}/\partial a_j^L$ and $f'(p^L)$ a vector containing $f'(p_j^L)$ respectively. The sign $\odot$ denotes the Hadamard product (elementwise multiplication).

The Error in Any Layer, Depending on the Error in the Next Layer

The title of the algorithm – the backpropagation algorithm – indicates that the propagation should go backwards and thus an equation for the errors in a certain layer expressed in terms of the errors in the next layer is desired. The expression for this is

$$\delta^l = (w^{l+1}\delta^{l+1}) \odot f'(p^l). \tag{3.20}$$

It can be worth noting that Equation (3.16) for the propagation in matrix form contains the transpose of the weight matrix, while the non-transposed matrix is used in the equation above. Since Equation (3.20) is a way of propagating the error backwards in the network and Equation (3.19) defines the error in the last layer, the error in any layer in the network can be computed.

The Derivative of the Cost with Respect to the Biases

To actually do some changes of the cost function a measure of how much the parameters of the network affects the cost is needed. Therefore, the derivative of the cost function with respect to any bias in the network is derived to be

$$\frac{\partial \tilde{C}}{\partial b_j^l} = \frac{\partial \tilde{C}}{\partial a_j^l}\frac{\partial a_j^l}{\partial p_j^l}\frac{\partial p_j^l}{\partial b_j^l} = \frac{\partial \tilde{C}}{\partial a_j^l} f'(p_j^l)\cdot 1 = \delta_j^l. \tag{3.21}$$

The Derivative of the Cost with Respect to the Weights

The last of the four equations will be the derivative of the cost function with respect to any weight in the network

$$\frac{\partial \tilde{C}}{\partial w_{ij}^l} = \frac{\partial \tilde{C}}{\partial a_j^l}\frac{\partial a_j^l}{\partial p_j^l}\frac{\partial p_j^l}{\partial w_{ij}^l} = \frac{\partial \tilde{C}}{\partial a_j^l} f'(p_j^l) a_i^{l-1} = a_i^{l-1}\delta_j^l. \tag{3.22}$$

It can be worth taking notice of the indices. To achieve the derivative, the activation of node i from layer $l-1$ is multiplied by the error of node j in layer l.

To summarize, the equations needed for the backpropagation algorithm are restated in the box below.

The equations needed for backpropagation:

$$\delta^L = \nabla_a \tilde{C} \odot f'(p^L)$$

$$\delta^l = (w^{l+1}\delta^{l+1}) \odot f'(p^l)$$

$$\frac{\partial \tilde{C}}{\partial b_j^l} = \delta_j^l$$

$$\frac{\partial \tilde{C}}{\partial w_{ij}^l} = a_i^{l-1}\delta_j^l$$

The Actual Algorithm

Now, all pieces that are needed to describe the actual algorithm are assembled. They only need to be put together. From the beginning to the end, one iteration of the backpropagation algorithm works as follows:

1. Input one or several inputs x to the network. Set the activation $a^1 = x$ for the first layer. Do this separately for the different inputs.

2. For each layer $l = 2, 3, ..., L$ compute the propagation p^l and the activation a^l — again separately for each input.

3. Compute the total network cost $\tilde{C}(w, b)$. Here, all different inputs will be taken into account.

4. Compute the output error δ^L.

5. For each layer $l = L - 1, L - 2, ..., 2$ compute the error δ^l.

6. The gradient of the cost function is given by all different $\partial\tilde{C}/\partial b_j^l$ and $\partial\tilde{C}/\partial w_{ij}^l$.

7. The gradient descent method can now be used to update the parameters of the network.

3.2 Convolutional Neural Networks

Convolutional neural networks (CNN:s) are a special kind of neural networks which are good to use when the input consists of data with spatial dependence, such as an image. The idea behind them is to put the computational effort where it gives the most profit.

An image is made up of several thousands of pixels which for fully connected layers means a large amount of weights. As an example, think of a 10 megapixel image. If the first hidden layer would contain as many nodes (each image pixel is a node) as the input layer that would mean $10^7 \cdot 10^7 = 10^{14}$ weights — and that is only for the first step.

Though, in an image the dependence is usually between nearby pixels and it is also that kind of information that is interesting. Actually, the human visual system works in a similar way [17]. Therefore, in a neural network for images, a certain node or pixel in layer i is set to be dependent of the pixels on that certain position, or nearby positions in the previous layer $i-1$. The number of pixels that are used is maybe 9, 16 or 25, instead of 10,000,000. That reduces the amount of computations a lot. That would mean that in the propagation function in Equation 3.1, most of the weights — except for a few — would be zero and the sum would only contain a low number of terms.

It has already been indicated that the shape of the image and the pixel orientations have importance and therefore the data is usually kept in the shape of the image, rather than stacked as a long column, as was the case in Figure 3.3. The data in the network is then referred to as feature maps, so one step in the network contains an input feature map, weights and biases and an output feature map. Since the nodes are arranged, so can the weights be, to easily touch upon the right pixels. That means that the propagation function (Equation 3.1) for a certain pixel in a certain layer can be computed by applying a kernel to the corresponding pixel in the previous layer.

Another thing that concerns networks of images is that the same features are usually interesting independently of the position in the image. If an edge is to be detected in the upper left corner, it is often desired to find an edge in the middle of the image as well. Futhermore, it is desirable that the network is somewhat robust to translations. If the network should detect a certain item in the image, the network should be robust enough to find that item even if it is moved a bit in the image. Therefore, the same operations should be applied all over the input feature map.

There are some different kinds of operations that can be performed, or rather some different types of layers, in a CNN. Some of those will now be presented.

3.2.1 Convolutional Layer

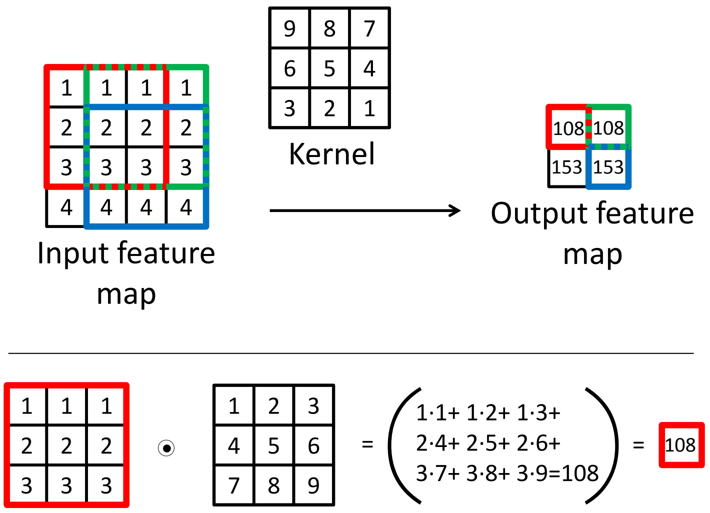

Figure 3.4: A convolutional step in a neural network, where the data is displayed as feature maps and the weights as a kernel. The lower part of the image shows how one pixel in the output feature map is computed using parts of the input feature map and the double flipped kernel. The multiplication sign in this figure denotes the Hadamard product (element wise multiplication).

The application of a kernel on the input feature map that was mentioned before is what is done in a convolutional layer. The same kernel is used everywhere in the input and this corresponds to 2D convolution of the input feature map with a kernel, see Figure 3.4. When 2D convolution is performed, the kernel to be applied is first flipped in the up-down-direction and then in the left-right-direction (or vice versa). Then, the kernel is “put on” the correct part of the feature map and element wise multiplication is performed. The final output value is then given my a summation of the achieved values.

The network step that is displayed in Figure 3.4 can also be seen in Figure 3.5. There it is displayed in the same way as the previous networks, with the input and output being column stacked.

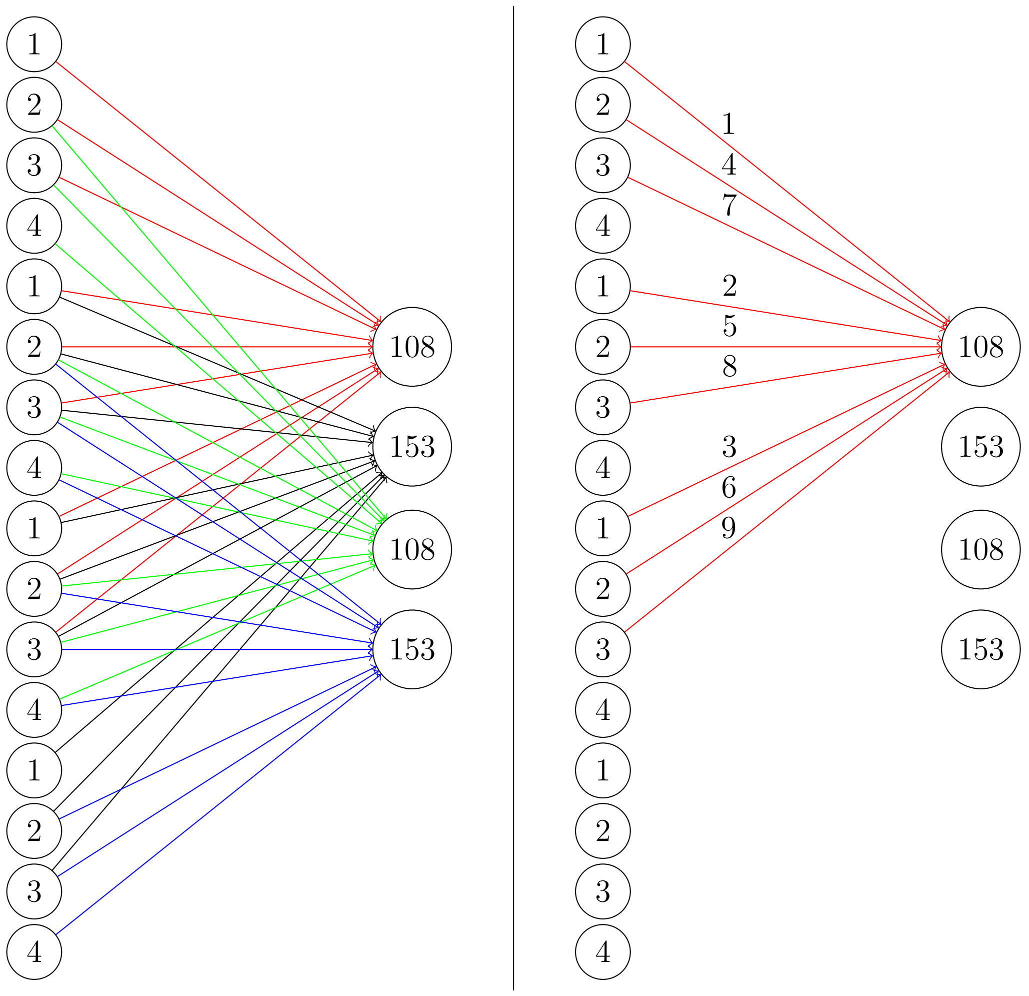

Figure 3.5: The image convolution in Figure 3.4 displayed as a regular network. To the left is the whole network, with the arrow colors corresponding to the colors of the squares in Figure 3.4 but with the weights left out. To the right is only the computations of one node in the output displayed — corresponding to the bottom part of Figure 3.4. Here the weights are written above the arrows. Note that the nodes in the two layers are not fully connected — that is that not all nodes in the first layer are connected to all nodes in the second.

Figure 3.4 only shows convolution with a single kernel, but in a convolutional step of a network, several kernels can be applied at once, each yielding one *channel* in the output feature map. To begin with, if the input feature map is an image of size $m_1 \times n_1$ and it contains d_1 channels, the input will be of size $m_1 \times n_1 \times d_1$. In an RGB image for example, $d_1 = 3$ and in a gray scale image $d_1 = 1$. Furthermore, each kernel that is applied to this input feature map has to have the size $m_2 \times n_2 \times d_1$, where m_2 and n_2 can be chosen independently of the size of the input, but the number of channels have to correspond. Then, having d_2 different kernels, results in a total four-dimensional collection of kernels of size $m_2 \times n_2 \times d_1 \times d_2$. Finally, as mentioned, the convolution of each $m_2 \times n_2 \times d_1$ kernel would result in one channel in the output feature map, yielding an output feature map with d_2 channels.

Concerning the size of the output feature map there are two different ways to perform 2D convolution. Either, the input feature map can be padded with zeroes such that the kernel can be applied to every pixel in the map (that is, the middle pixel of the kernel can be "laid on" every pixel of the input). That would result in an output size of $m_1 \times n_1 \times d_2$. The other option is to not use zero padding and then the kernel cannot be applied to the outermost pixels of the input. Depending on the size of the kernel, the output size would then be $(m_1 - m_2 + 1) \times (n_1 - n_2 + 1) \times d_2$. The convolution described in Figure 3.4 and 3.5 is done without zero padding and that is what will be used when referring to convolution in this report.

Often in neural network context the kernels are referred to as filters and the collection of all filters in a certain layer is called a filter bank.

3.2.2 Max-Pooling Layer

The max-pooling layer is a non-linear subsampling layer which is applied separately to each channel of the input. Once again the idea that the network should be robust to small translations is applied. To perform the max-pooling a window size α and a stride length β are required. A window of $\alpha \times \alpha$ nodes from the input feature map yields one node in the output feature map. This node will be the maximum of the numbers found inside the window. The stride length β decides how close the different windows should be. If $\beta = \alpha$, then the whole input feature map will be divided into $\alpha \times \alpha$ sized parts, resulting in an output feature map which in both dimensions has decreased in size by a factor β. It is common to use $\alpha = \beta = 2$. An example of max-pooling with window size two and stride length one and two can be seen in Figure 3.6. In the case with stride length two, the windows are non-overlapping and thus each node in the output feature map is created by different nodes from the input feature map. Since the operator is applied separately to each channel, the number of channels of the output feature map will be equal to the number of channels of the input feature map.

The max-pooling layer also contributes with some spatial invariance to the network, such that it does not matter exactly where a feature is found in the image, but more that it is present.

3.2.3 ReLU Layer

The ReLU layer is performed by taking each pixel of the input feature map as input to the ReLU function in Equation (3.7) to get the output feature map. Thus, the size of the output from the ReLU layer is exactly the same as the size of the input.

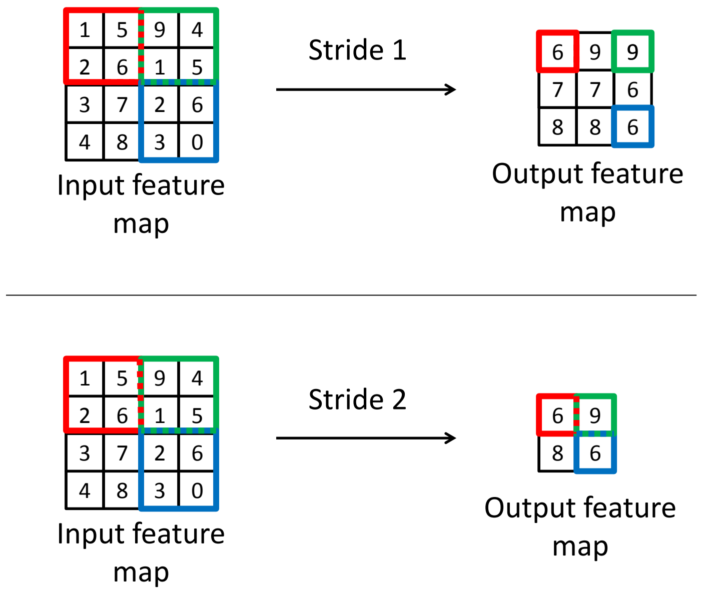

Figure 3.6: A max-pooling step of a CNN. The upper part of the image shows max-pooling with window size two and stride length one and the lower part shows max-pooling with window size two and stride length two. In the lower example the windows are non-overlapping.

3.2.4 Softmax Loss Layer

Finally, the network needs a layer than can perform classification and compute the cost — from Equation (3.10) — for the network. By studying Equations (3.9) and (3.10) it is clear that these take the same form no matter which loss function that is used. The example given in Equation (3.8) is a simple loss function which is commonly used. Though, for the networks that have been implemented in this thesis, another loss function has been used, namely the softmax loss function. The softmax loss is a combination of the softmax function and the logarithmic loss function [15].

The Softmax Function

The softmax function is a normalizing function

$$y_{ijk'} = \frac{e^{x_{ijk'}}}{\sum_{k=1}^{d} e^{x_{ijk}}}. \tag{3.23}$$

Here, the first two indices, i and j, refer to the pixel and the third index, k or k', to the layer or channel. To explain Equation (3.23) a bit further, the exponential of each pixel in a layer k' is normed by the sum of the exponentials of that pixel value in all d layers. In the convolutional neural network this will be performed for all pixels in all layers. The result will be that all values will be between 0 and 1. Furthermore, for a certain pixel a summation over all layers would yield one, as should be the case after applying a normalizing function.

The Logarithmic Loss Function

The logarithmic loss function is what actually replaces the loss function in Equation (3.8). The input to the function is supposed to be between 0 and 1 and in the softmax loss layer the logarithmic loss function is applied after the softmax function. The logarithmic loss is defined as

$$y_{ij} = -log(x_{ijc_{ij}}), \tag{3.24}$$

with c_{ij} being the index of the ground truth class at pixel (i, j). This means that if the network predicts that pixel to be of the correct class, the value $x_{ijc_{ij}}$ will be close to one and the loss y_{ij} will be small. On the other hand, if the pixel is wrongly classified, $x_{ijc_{ij}}$ will contain a low value (low probability of being of the correct class c_{ij}) and the loss will be large. In the case of the softmax loss layer, the input to the logarithmic loss function will be the output of the softmax function.

In the earlier loss function — Equation (3.8) — a summation was performed over all output neurons. That would correspond to adding a double sum in front of Equation (3.24), summing over all i:s and all j:s. This is done if different pixels in or different parts of an input image belong different classes. Though, convolutional neural networks are often used to classify a whole image. In that case no summation is needed, since the classification output will be of size $1 \times 1 \times \gamma$, where γ is the number of possible classes. Of course, for this to match up, the network needs to be adjusted such that the input to the classifying softmax loss layer is of that size as well.

3.3 Network Topology

Within this thesis, two different network topologies have been implemented — a simple neural network and a network with a directed acyclic graph structure.

3.3.1 Simple Neural Network

The simple neural network is — just as it sounds — a simple kind of network which represents networks with a linear topology. This means that the network is a straight chain of layers, where the output from one layer is the input to the next and so forth. Compared to the directed acyclig graph network explained below, the simple network is built as a graph. [18]

3.3.2 Directed Acyclic Graph Network

The directed acyclic graph (DAG) network can represent more complex networks than a simple network can. The structure is object oriented, where layers and variables are viewed as different objects. Layers are connected to variables, and variables are connected to layers. In addition to one or several variables, each layer can also have parameters as inputs. To reconnect to previous explanations, recall Equation (3.15). The computation of the activation is performed "in" the layer, which is defined by the function $f(x)$. The value a^{l-1} is the input variable while the resulting value a^l will be the output variable. The bias b^l and weights w^l are examples of parameters.

The main point of the DAG network is that the structure does not have to consist of a straight chain of layers. Thus, one layer can have two different input parameters, which are outputs from two different previous layers. Because of this, it is a good idea to implement the network as a graph. Futhermore, the acyclic property makes the network easier to work with. If the network would have cyclic properties, this would yield much more complicated computations, e.g. when the parameters are to be updated. Still, the DAG structure is more complicated, but also more variable, than the simple structure explained above. [18]

4 Deep Neural Networks in Practice

4.1 Software

All the implementations done within this thesis are performed in `Matlab`. For the implementations of convolutional neural networks the MatConvNet toolbox has been widely used. This is a toolbox created by the Oxford Visual Geometry Group and it can be found in [18]. It contains many of the functions that can be desirable when building and working with neural networks. There are different wrappers to work with either simple neural network or DAG neural network and all the layers that have been described here, as well as the actual training process, are already implemented. Thus, only smaller changes have been made to suit the specific problem.

The tests have been performed on a computer with an Intel Core i5-6400 processor with 8GB RAM memory-

4.2 The Data

To train the network to perform Gleason classification, digital microscopic images of prostate biopsies have been used. All samples have been stained with haematoxylin-eosin (H&E) before the images were taken. The biopsy samples come from two different datasets, one which was supplied by Beaumont Hospital, Dublin, Ireland and one which was provided by PathXL, Belfast [6]. The images were previously used in [6].

Furthermore, the images are cut-outs of these biopsy samples, such that each image only contains one class — that is benign, Gleason 3, Gleason 4 or Gleason 5 tissue. For this reason, the images consist of various numbers of pixels. In some cases a small part of the image — like a corner — does not contain any sample. Even if all images are stained in the same way the colors differ somewhat because of variations such as light, the staining being performed by different people etc. [6]. Some examples of images can be seen in Figure 4.1. These can also be compared with the Gleason grading system in Figure 2.2. In total the original two image sets contain 213 images, 52 each of benign, Gleason 3 and Gleason 4 and 57 of Gleason 5.

4.2.1 Ethical Aspects

The images used for this project are real samples of prostate biopsies. Thus, they come from real people. There is no way of identifying from whom the samples are taken and this project only has medical aims. No matter, there are ethical aspects as soon as

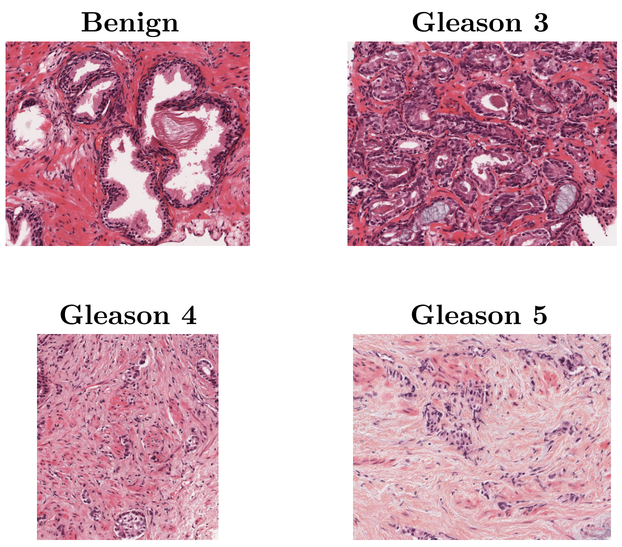

Figure 4.1: Examples of the images that were used to train the networks. The figure shows one image of each class. These can also be compared to the Gleason scale given in Figure 2.2. Notice how the glandular structure is gradually lost with increasing Gleason grade.

human beings are involved. Because of this the project is approved by the Regional Ethics committee at Lund University with the ethical permit number 2013/400 [19].

4.2.2 The Keys

One main idea with this thesis was to see whether it could have any impact to use a “key” in the middle of the network, consisting of a tissue classification. These keys have been created by hand using Matlab and the computer mouse. There are only keys for the benign images. The key for each image consists of a number of masks. The masks are created to contain certain parts of the prostate, namely four different *classes* or *parts*.

The first part is called *stroma* and simply consists of that. The second class is called *glandWallCells* but is actually not all of the cells, but only the nuclei of the cells making up the gland wall. In the images these are dark purple round spots which are collected around the glands, see Figure 2.1. The third part, *glandWallPurple* constitutes of the rest of the cells in the gland wall, where there are no nuclei. This part is inside the glandWallCells and outside the glandular lumen and has a lighter color than the nuclei. The fourth and last part is called *whiteParts*. This is a combination of the glandular lumens and the parts of the image that does not contain any sample. Since the background (behind the samples) is white, the lumen and the background are considered to be very similar — actually too similar to be different classes when the surroundings are not included. An example of a segmentation image key can be seen in Figure 4.2.

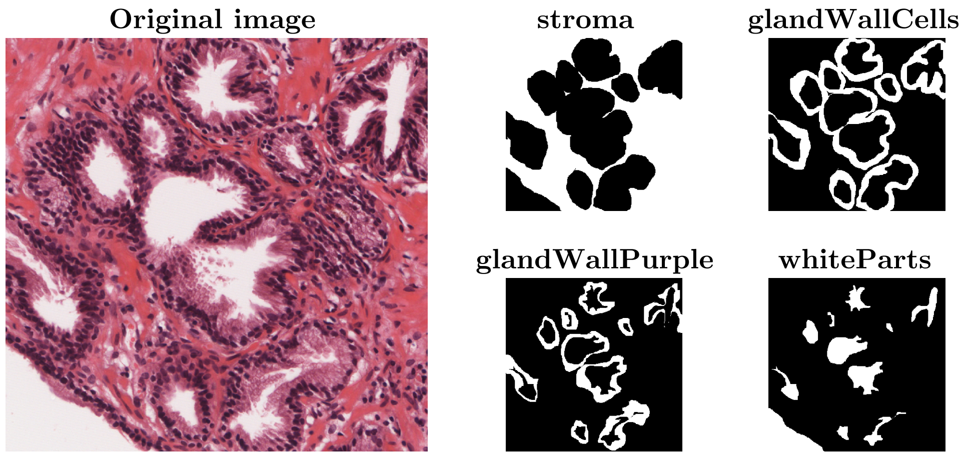

Figure 4.2: One example of the keys used in this project. To the left is the original image, while the segmentation keys can be seen to the right. White indicates presence of the certain class, while black indicates absence. The four classes are stroma, glandWallCells, glandWallPurple and whiteParts.

There are 52 benign images in the dataset, but keys have only been created for 20 of these. It should also be mentioned that the key masks are created by an engineering student with quite low knowledge of the prostate anatomy and not by a physician.

4.3 How to Build a Network

Even if the base for the implemented networks is taken from MatConvNet the architecture of the network is variable. Some different network architectures have been tried, but most were created on gut feeling and at the end a fairly methodical architecture, presented below, was chosen. The reasoning below only concerns simple, straight networks.

4.3.1 Connecting the Layers

The networks have a base of layer *triplets*. Each triplet is made up from one convolutional layer, one ReLU layer and one max-pooling layer. This means that each such part of the networks consists of a linear operation, a non-linear operation and some downsampling. This is a common way of building a network, even if the choice of operations may vary. To begin with, if there are two convolutional layers in a row, these two linear operations might as well be replaced by only one, since the convolution is a linear operation. After the convolutional layer an activation function — here the ReLU function — is applied.

This can be compared to biological neurons, where the biological "activation function" will determine whether that neuron should fire or not [2]. Then finally, the downsampling is applied. This is done to decrease the computational efforts, the size of the feature map and prepare for classification [12].

4.3.2 Kernel Sizes and the Number of Filters

For the max-pooling a window size of 2×2 and a stride 2 has been used throughout this thesis. This means that except for the small decrease of feature map size that occurs after the convolutional layers, the feature map size will be halved — from $m \times m$ to $m/2 \times m/2$, so the number of pixels will be decreased to a quarter — with each layer triplet. The number of filters — and thus output channels from the convolutional layers — have been doubled in each triplet. It is common that the number of filers are increased throughout the network. Since the feature map at the same time size is decreased with each triplet, the computational effort will not be increased due to this. The number of filters in the beginning of the networks are set to be 20, which would give the filter bank at the first convolutional layer a size $5 \times 5 \times 3 \times 20$ and at the second convolutional layer a size $5 \times 5 \times 20 \times 40$. This is of course if the kernel size is 5×5, which has mainly been the case.

In the beginning, when setting up the networks, the weights have been initialized as random normally distributed while the biases have been initialized to zero. The mean of the weights has been kept to zero and the standard deviation f has been varied. All the convolutional layers have been applied without zero padding.

4.3.3 The End of the Network

Depending on the size of the network input image a number of triplets have been put together until the output feature map is of size $1 \times 1 \times d$, where d is the number of layers, in this case $d = 20 \cdot 2^{\chi-1}$, with χ being the number of triplets. Next, one fully connected layer was added, in which the layers are doubled. That the layer is fully connected means that there are connections from all nodes in the first layer, to all nodes in the second. This layer was then, as was explained earlier, followed by a ReLU layer. Then, no pooling layer is needed, since each layer only consists of a single pixel. It is common that convolutional neural networks are created both with and without these layers [12]. After this, to get one output per possible image class, one last — also fully connected — convolutional layer was added to the network. That filter bank is of size $1 \times 1 \times d \times \gamma$, with γ being the number of possible classes. That is, each filter will have the size $1 \times 1 \times d$ and there are γ different filters in the bank. Finally the softmax loss layer was added to finish the network.

The softmax loss layer gives a total network cost such that the network can be trained. If the network is already trained and should be used solely for classification, the softmax loss layer can be substituted by a layer which applies only the softmax operation. That then gives an output with class probabilities and the input would be classified as belonging to the class corresponding to the highest number in the output.

4.4 Choice of Parameters

When a network is set up and trained, there are some hyper-parameters that needs to be chosen. Examples of these are the learning rate η, the momentum variable μ, the batch size for the stochastic gradient descent and the number of epochs to train the network. Within this thesis the focus has mainly been to explore neural networks and implement an architecture that has not earlier been applied to the images in the project, rather than systematic optimization of these hyper-parameters. Thus, the hyper-parameters have been tuned by hand, by trying some out and then visually comparing the results — more specifically the error rates for the classification — with the results for some other hyper-parameter values. Most of the first try values for this have been inspired by the values used in the MatConvNet CNN practical tutorial [15] and after that the values have been adjusted as described.

4.4.1 Adaptive Learning Rate

Instead of choosing a single value for the learning rate η, another approach can be taken, namely an adaptive learning rate. That means that the learning rate is adjusted through the network. Usually this means that η is decreased for later epochs, such that the learning rate is lower and the network can be more finely adjusted when the cost function has decreased moderately. Though, in which way and what it depends on when the learning rate is adjusted can differ. Within this thesis, the adaptive learning rate has been implemented as a decreasing step function.

4.4.2 Number of Training Epochs

The network training should be run until it is saturated. This is considered to be when the objective function for the training data has stopped decreasing. Usually it starts to oscillate somewhat at this point, because the network tries to adjust for improvements, but no proper ones are achieved. When this occurs, the objective for the validation set has sometimes already increased a bit from its minimum, while the errors have stabilized. The number of epochs is dependent on the choice of the other hyper-parameters, e.g. if the learning rate is decreased, the number of training epochs will probably have to be increased.

4.5 Reduce Overfitting

Overtraining — or overfitting — is when the network is trained to perform very well on the training data but is adapted just to that data and is thus not general enough. Then the network will perform badly on other data which is not part of the training set. Overfitting can be a problem in artificial neural networks and so has also been the case in this project. There are some things that can be done to reduce overfitting.

4.5.1 Increased Amount of Data

The more data that is used, the smaller is the risk of reaching overfitting. When it comes to the prostate images, there is quite a small amount of images and these should be used both for training and validation. Though, one way to decrease overfitting would be to collect more data.

If it is hard or not possible to collect new images the possessed images can be used to create new, artifical data. This can be done in several ways. One way to do so is to add noise to the original image, different noise in each training epoch. Then the image will not look the same throughout the training and thus the risk of overfitting is decreased. This method has been implemented in this thesis.

Another way to add artificial data is to mirror and rotate the images. For simplicity, assume that an image is only rotated 90, 180 and 270 degrees to avoid the problem of corners sticking out. Then the image is also mirrored, both in the left-right and the up-down direction. That would give six images to train on instead of one. In the same manner, the images could be skewed, stretched and so forth. This would yield a larger variation och more images. In the classification problem, the rotation and mirroring has been used.

4.5.2 Dropout

Dropout is another method used to prevent overtraining. When using dropout, some nodes of the hidden layers are "shut off" during training. The input and output nodes are left untouched. Then, the network is run, both forward and backward, with the nodes that are left, while the shut off nodes are not allowed to take part in the training. After adjusting the used weights a new set of nodes can be chosen and a new training step performed, and so forth. Thus, all nodes are likely to take part in the training, but not all the time. The proportion of hidden nodes φ that are shut off can be chosen and depending on that the trained weights need to be adjusted, since more information will come when all nodes are on, than when only parts of them are [16].

When the network has been trained all nodes and weights are used for classification, so it is only for the training part the dropout is used.

The dropout as it is described above has been applied in the implementations. Things that can be varied are the proportion of shut off nodes φ but also the number of layers that dropout is applied to. Though, dropout in generally applied to fully connected layers. This means that in the networks used here there have only been a few places to add dropout, namely at the last convolutinal layers. Hence, dropout has been applied at these places.

5 Implementations

To get a feeling for the architecture of the classifying DAG network, a brief overview is given in Figure 5.1. In the figure, each block consists of one or several network layers — and in most cases one or more layer triplets. The input is the image to be classified and output1 is the classification of that image — that is output1 will tell whether the image is either benign, Gleason 3, Gleason 4 or Gleason 5. The second output, output2, is the segmentation of the tissue parts — stroma, glandWallCells, glandWallPurple or whiteParts. To perform training of the network, a softmax loss layer is applied to the true class label together with output1. In a similar way, a middle key image with classifications, similar to the one in Figure 4.2, is added to output2 and then the backpropagation algorithm can be used. The term *middle* key simply refers to the fact that this label is placed in the "middle" of the network (remember that label and key is the same thing).

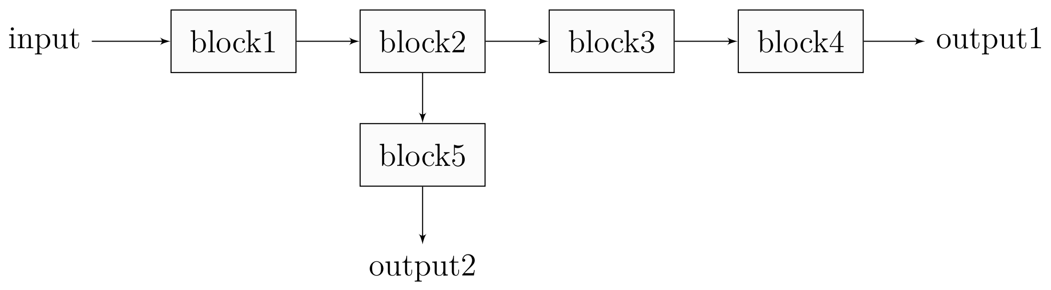

Figure 5.1: A brief overview of the architecture of the classifying network. The network is built up as a straight, simple network, with one outgoing branch. Concerning the variables, input is the image to be classified, output1 is the predicted class or Gleason grade of the image and output2 is the segmentation of the different tissue parts. Each block can contain one or several network layers.

As was explained in Chapter 2 the glandular structures are lost when cancer develops. The glands become smaller and the lumens more narrow. The idea with the middle key was that if the network already by block2 (see Figure 3.1) could have an idea of where in the image and in what amounts the different tissue parts are present that could help the final tissue classification.

5.1 The Segmentation Problem

As has been mentioned several times, the segmentation network is a straight network with simple topology. It could be seen as the part of the network in Figure 5.1 consisting of the input, block1, block2, block5 and output2.

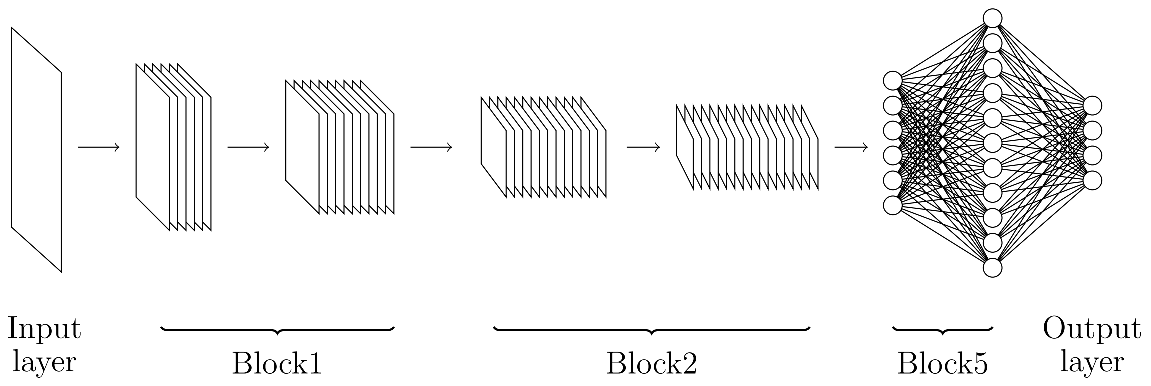

Figure 5.2: The segmentation network. The layer stacks and column stacks are the different feature maps. Between each layer stack there is a layer triplet consisting of a convolutional layer, a ReLU layer and a max-pooling layer. Block5 consist of two fully connected convolutional layers with attached ReLU layers. There, dropout is applied as well.

A more specific architecture of the network can be seen in Figure 5.2. The network begins with four layer triplets, each consisting of a convolutional layer, a ReLU layer and a max-pooling layer. All of the convolutional layers have kernel size 5×5 and they have 20, 40, 80 and 160 different filters respectively. The max-pooling layers have both kernel size and stride 2.

These triplets are followed by two fully connected convolutional layers, each with a consecutive ReLU layer and with dropout applied. The first convolutional layer has a filter bank of size $1 \times 1 \times 160 \times 320$ — that is the number of layers are doubled, just like before — while the second one has a filter bank of size $1 \times 1 \times 320 \times 4$ to get the correct number of inputs, one per possible output class. The applied dropout has a *dropout rate* φ which means that a part φ of the nodes are allowed to continue through the network, while $1 - \varphi$ are shut off, each time dropout is applied.

Finally, after these layers, the network has a softmax loss layer to classify and get a network cost to use for training.

5.1.1 The Input Patches

The output pixels from the network just described get a receptive field size of 76×76. That means that each pixel in the output is dependent of a square of 76×76 pixels in the input. Thus, the input to the network should be of that size, for the input to be classified only with one class.

All patches that are used as input during the training are cut out from benign images. In total the 20 images that have a key have been used. As explained earlier, in Chapter 4.2.2, the patches are of four different classes — stroma, glandWallCells, glandWallPurple and whiteParts. The keys (Figure 4.2) work as a "maps" to decide where the different classes can be cut. Though, for some of the classes the coherent areas of pixels are quite narrow. To make it possible to get more patches of those classes, not all of the cut out patch has to contain the specific class. Primarily, patches only containing that class are cut, but if there are not enough possible patches, 10% of the patch is allowed to contain other classes. If there still are not enough, 20% are allowed to be from another class and so forth. The maximum allowance is set to 50%, but this has only been used in a few occasions.

The 20 benign images have all been used to generate input data. The images have been divided in four sets of five images each. These four sets have been used to perform four-fold cross-validation, with one set at a time being used as validation set, while the other three have been used as training set. For each set, 1000 patches of each class were cut. This means that a total of 16,000 input patches were used, of which 12,000 were in the training set each time. From which of the large, benign images each patch was cut was chosen randomly within the set. Where in the large image the patches were cut out was also chosen randomly, within the correct class. A few examples of patches can be seen in Figure 5.3.

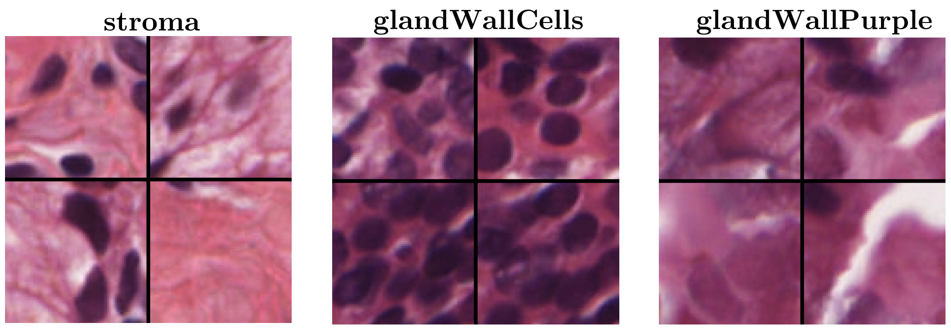

Figure 5.3: Four examples of input patches for stroma (to the left), glandWallCells (in the middle) and glandWallPurple (to the right). No patches of whiteParts are included, since they are mainly white. Each patch is 76×76 pixels.

Pre-Processing

Before the patches were cut and run through the network the image mean was removed. When this is done, the image specific intensity matters less, and the actual variations in the picture and the tissue becomes more important. In this thesis, the mean has been removed layerwise from each image. This pre-processing was been performed on all images, both those used for training and validation.

Furthermore, some random, zero mean, Gaussian noise was added to the training patches. Different noise was added to each image and this was remodeled for each epoch. This was done to bring some variation to the input data and make overfitting less probable.

5.1.2 Training the Network

	Setting 1	**Setting 2**	**Setting 3**	**Setting 4**	**Setting 5**
f	1/100	1/100	1/100	1/100	1/100
τ	100	100	100	100	100
φ	0.3	0.3	0.5	0.3	0.5
η	$7 \cdot 10^{-4}$	adaptive1	$7 \cdot 10^{-4}$	$7 \cdot 10^{-4}$	$7 \cdot 10^{-4}$
μ	0.9	0.9	0.9	0.75	0.95
λ	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	$7 \cdot 10^{-4}$

Table 5.1: The different hyper-parameter combinations that were tried. Here, f is the standard deviation of the initialized weights, φ is the dropout rate, τ is the batch size, η is the learning rate, μ is the momentum variable and λ is the weight decay parameter. Furthermore, adaptive1$= 8 \cdot 10^{-4}\theta(x) - 3 \cdot 10^{-4}\theta(x-10) - 4 \cdot 10^{-4}\theta(x-25) - 5 \cdot 10^{-5}\theta(x-40) - 4 \cdot 10^{-5}\theta(x-60)$.

All the training has been performed using stochastic gradient descent with a batch size τ. The learning rate η was in some cases constant through all epochs while it was sometimes adaptive. In the adaptive case the learning rate was, as mentioned, a step function depending on the current epoch, $\theta(epoch)$. The number of patches used for the training and testing was constant, as described above, and even if they were selected randomly the sets have been the same through all networks, to make possible for a fair comparison. Furthermore, other hyper-parameters are the standard deviation for the initialization of the networks weights f, the dropout rate φ, the momentum variable μ and the regularisation or weight decay parameter λ. A few different parameter settings that were tried are presented in Table 5.1. In one of the tried settings, the learning rate was adaptive, such that $\eta = \text{adaptive1} = 8 \cdot 10^{-4}\theta(x) - 3 \cdot 10^{-4}\theta(x-10) - 4 \cdot 10^{-4}\theta(x-25) - 5 \cdot 10^{-5}\theta(x-40) - 4 \cdot 10^{-5}\theta(x-60)$. The training was run for 150 epochs for all different settings.

5.1.3 The Evaluation

As has been mentioned several times, the network has been trained to minimize the objective function. To compare different hyper-parameter setting and runs to each other, the patch error has been used. For each training epoch, the rate of erroneously classified patches has been computed. The lower error rate, the better. Both the objective and the patch error were plotted as functions of the epoch, to clarify the results.

5.1.4 The Final Training

To achieve a trained segmentation network to use in the classification network, setting 5 from Table 5.1 was used. For the final training, no images were saved for validation, but all images were used for training. That means that the final training was performed on 16,000 training patches. Since the cross validation had already been performed and an error rate for that had been computed it was better to use as much data as possible to achieve a good network.

This final network can be used to perform a kind of segmentation on a whole image. Though, the output from the network will be a smaller image than the input, with each pixel having a receptive field 76×76, but it will resemble the input image with the tissue structures being caught. Desirably, the different output channels should be segmentations of the different classes in the image. The network has been applied on both images of benign and cancerous tissue, since this will be the case in the classification network. These were images that were not included in the set used for training.

5.2 The Classification Problem

The classification network was built as was briefly explained in Figure 5.1. To connect this to the segmentation network, the first four layer triplets of that are included in block1 and block2, while the fully connected layers and the loss layer belong to block5. Thus, to develop the classifying network block3 and block4 were added to the segmentation network, resulting in a network which needs the DAG architecture for implementations.

The details for the network architecture are illustrated in Figure 5.4. The classifying network starts with four layer triplets with the same properties as explained in the previous section. These are followed by another two layer triplets. The convolutional layers still have a kernel size 5×5 and the first of the two has 320 filters, while the second has 640. The pooling is again performed with kernel size 2 and stride 2. These two triplets belong to block3 in Figure 5.1, such that block1, block2 and block3 all contain two triplets each.

Then remains block4, which is partly equivalent to block5 in the segmentation network. At the end of the classifying network, after the six layer triplets, there are two more convolutional layers, one with a filter bank of size $1 \times 1 \times 640 \times 1280$ and one with a filter bank of size $1 \times 1 \times 1280 \times 4$. The convolutional layers are followed by ReLU layers and dropout is applied. Finally, there is a softmax and a softmax loss layer, to yield the output prediction and to make possible for backpropagation.

5.2.1 The Input Patches

The output pixels from block4 in the classifying network have a receptive field size 316×316 and thus, this has been the size of the input to the network. To cut input patches to the network, 10 images of each class — benign, Gleason 3, Gleason 4 and Gleason 5 — have been used. The benign images are taken from those that have a corresponding key. Since all images are of different sizes, they yield different numbers

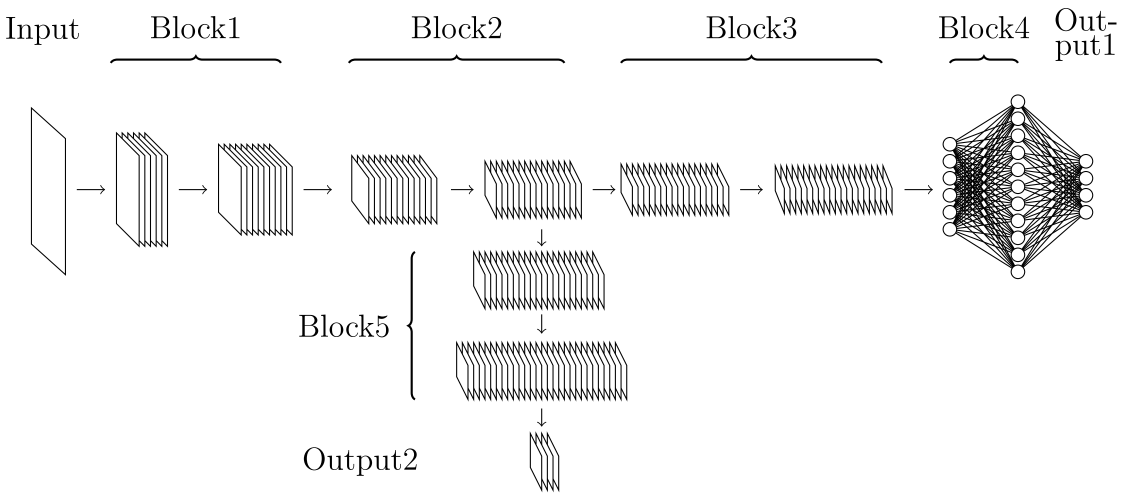

Figure 5.4: The classification network. The layer stacks and column stacks are the different feature maps. Between each layer stack there is a layer triplet consisting of a convolutional layer, a ReLU layer and a max-pooling layer. Block4 and Block5 consist of two fully connected convolutional layers with attached ReLU layers. For these, dropout is applied. In comparison to Block4, the full connections in Block5 are denoted with a single arrow for a clearer visualization. Though, they are fully connected and the feature map sizes are constant.

of patches. From each image, as many patches as possible have been cut, without the patches overlapping. If the image dimensions were not divisible with 316, the edges of the image were not included in any patch. The smallest image gave 2 patches, while the largest gave 60 patches. In total 1074 patches were cut, 344 of benign tissue, 239 of Gleason 3, 220 of Gleason 4 and 271 of Gleason 5. At this stage, the mean of each patch was removed, in the same manner as was done with the whole images for the segmentation network.

After the patches were cut, each patch was modified to give more, artificial, input data. Except from using each patch as it is, another one with zero mean Gaussian noise added was used. Then, the original patch was mirrored – both in the left-right direction and the up-down direction. Furthermore it was also rotated 90, 270 and 360 degrees, just as described in Chapter 4.5.1. This means that each cut patch gave seven inputs to the classifying network.

One difference from the segmentation network is that a re-cut of the patches in this case always would yield the same patches, except for the noisy one, since there otherwise is no randomness.

5.2.2 The Middle Keys

Since the input to the whole, classification network is of size 316×316, output2 in Figure 5.1 will have the size 16×16 (where each of the $16 \cdot 16$ pixels has a receptive field 76×76).

That means that the middle keys need to have that size. The class labels simply contain a 1×4 vector, with a 1 on the place corresponding to the correct class. It could be seen as if there is one channel per class, but that only one of those channels can be non-zero. This is called a categorical label [18]. Instead of this, the middle keys contain several pixels and also several channels, one for each segmentation class. That means that the middle keys also have four channels — one for stroma, one for glandWallCells, one for glandWallPurple and one for whiteParts. Though, each channel represents whether the class of that specific channel is present or not, like attributes [18]. A 1 for pixel i, j in channel k means that attribute/class k is present in the receptive field of pixel i, j. In the same way, -1 means that the attibute is not present. Compared to the categorical label, more than one class can be present, when attributes are used.

To create one of the pixels in the middle key, the 76×76 pixels which that pixel origins from were cut out of the original key, explained in Chapter 4.2.2. Then the cut out key part was still categorical. After that the ratio of each class or attribute in that cut out part was computed. All attributes that had a ratio above a certain threshold δ were considered to be present and were marked with a 1 in the middle key, while those that had a ratio below δ got a -1. Then all the middle key pixels were created in the same manner.

5.2.3 The Logistic Logarithmic Loss

In the segmentation network, at the end of block5, the softmax loss function was used for predictions and backpropagation. Though, this function cannot be used when the label, or in this case middle key, is not categorical, which for the classification network is the case at output2. Therefore, another, similar loss function and network layer has been used, namely the logistic log loss function

$$l(x, c) = \log(1 + \mathrm{e}^{-cx}). \tag{5.1}$$

In the equation above, c is the attribute, that is -1 or $+1$ depending on whether the attribute is present or not. The value x is the prediction score from the network. It is clear that the value of the loss function will be lower if c and x have the same sign. The logistic log loss function is similar to the softmax log function, but for attributes instead of categorical labels [18].

The objective function for the classification network looks the same as for the segmentation network, see Equation (3.10).

The Middle Keys for the Cancerous Tissue

There are only keys for the benign images (Chapter 4.2.2), but to run the classifying network there needed to be keys for all images that patches were taken from. When cancer develops in the prostate, the glandular structures are gradually lost. At a glance, the prostate tissue that contains cancer looks quite similar to stroma, see Figure 4.1. Because of that, the keys for the images with Gleason 3, Gleason 4 and Gleason 5 were approximated to consist only of the key class stroma.

5.2.4 First Training Approach

The training of the network has been approached in three different ways. In all cases, the training has been performed for a parameter setting which allows the objectives to change. As soon as such a setting has been found, no more optimization or comparison between parameter settings has been performed.

In the first case, the network was initialized in the same way as the segmentation network was, with random weights, and the network was trained from scratch. The network hyper-parameters that were used are presented in Table 5.2.

Table 5.2: The hyper-parameter setting that was used for the first approach of the training of the classification network. Here, f is the standard deviation of the initialized weights, δ is the threshold for creating the middle key, φ is the dropout rate, τ is the batch size, η is the learning rate, μ is the momentum variable and λ is the weight decay parameter. Furthermore, adaptive2= $2 \cdot 10^{-3}\theta(x) - 1 \cdot 10^{-3}\theta(x-12) - 5 \cdot 10^{-4}\theta(x-62) - 3 \cdot 10^{-4}\theta(x-112) - 1 \cdot 10^{-4}\theta(x-162) - 3 \cdot 10^{-5}\theta(x-235) - 3 \cdot 10^{-5}\theta(x-350)$

f	δ	τ	φ	η	μ	λ
1/100	0.1	100	0.3	adaptive2	0.9	$5 \cdot 10^{-4}$

5.2.5 Second Training Approach

In the second training approach, the trained segmentation network was used to initialize the weights and biases in block1, block2 and block5. The parameters in block3 and block4 were initialized as in the first training approach and the whole network was trained from there. All parameters could be changed within the network training. The used hyper-parameters can be seen in Table 5.3.

Table 5.3: The hyper-parameter setting that was used for the second training approach of the classification network. An explanation of the variables can be seen in the caption of Table 5.2. Here, adaptive3= $1 \cdot 10^{-3}\theta(x) - 5 \cdot 10^{-4}\theta(x-7) - 3 \cdot 10^{-4}\theta(x-14) - 11 \cdot 10^{-5}\theta(x-84)$.

f	δ	τ	φ	η	μ	λ
1/200	0.1	200	0.3	adaptive3	0.9	$5 \cdot 10^{-4}$

5.2.6 Third Training Approach

In the last way of training, the parameters in block1, block2 and block5 were set to be constant. The parameter values were the ones from the trained segmentation network.

Thus, they were not allowed to change during training. The only values that could change were those in block3 and block4, which again were initialized in the same way as before. In this case, objective2 did thus not change at all, and has not been included in the plots. The hyper-parameters that were used for training can be found in Table 5.4

Table 5.4: The hyper-parameter setting that was used for the third training approach for the classification network. An explanation of the variables can be seen in the caption of Table 5.2. Here, adaptive4= $1 \cdot 10^{-2}\theta(x) - 6 \cdot 10^{-3}\theta(x-5) - 2 \cdot 10^{-3}\theta(x-20) - 1 \cdot 10^{-3}\theta(x-40) - 4 \cdot 10^{-4}\theta(x-50) - 4 \cdot 10^{-4}\theta(x-70) - 6 \cdot 10^{-5}\theta(x-100)$.

f	δ	τ	φ	η	μ	λ
1/200	0.1	200	0.3	adaptive4	0.9	$5 \cdot 10^{-4}$

5.2.7 The Evaluation

As for the segmentation network, the classification network has been trained to minimize the objectives. Though, in this case there have been two of them, objective1 at the end of block4 and objective2 at the end of block5. This is taken care of by the backpropagation function, where some of the layer errors, see Equation (3.20), will depend on more than one consecutive layer. The layer numbering is not as simple, but the equations in Chapter 3.1.4 can be adjusted to suit the non-straight DAG nets as well and this is implemented in the MatConvNet package.

Furthermore the patch error has been computed in the same way as in the previous case. Though, when it comes to classifying the whole prostate images, the prediction of the patches is not as interesting as the prediction of the whole images. This prediction has been computed in two different ways.

In both image prediction approaches the patches get to "vote" about which class the image belongs to, but it is done in different ways. In the first approach, each patch votes with a 1 for the class that the patch is predicted as, and 0:s for all other classes. The class with the highest vote wins and the image is classified as that class.

In the second approach, the patches does not vote with 1:s and 0:s, but with the probabilities. For each image, the outputs from the softmax layer for all different patches are added. Thus, one patch might vote with probability 0.7 for one class and 0.3 for another, and this is taken into account for the image classification. Again, the image is classified to belong to the class with the highest vote after all patches have voted.

Finally, an image error rate has been computed for each epoch, in the same manner as the patch error rate. Since there have been two ways to predict the image class, there have also been two image errors, imageError1 computed by the first prediction approach and imageError2 computed by the second approach. To visualize the results, the two image errors, the two objectives and the patch error have been plotted over all epochs. For the classification network, no error rate has been computed for output2, at the end of block5.

6 Results

6.1 The Segmentation Problem

6.1.1 Training Statistics

The achieved objectives and errors from setting 2 and setting 5 can be seen in Figure 6.1 and 6.2. The plots for the other settings look similar and are therefore not included. In some cases the objective and the error for the validation set are much lower than the ones for the training set. This is due to the applied dropout. The dropout is applied to two fully connected layers, each time with a dropout rate $\varphi = 0.3$. That means that many nodes have been shut off during training, which has made it hard for the network to do correct predictions. When the validation set was run through the network, no dropout was applied and the results are better. Though, if the training set is run through the network without dropout, the objective and error are even lower than for the validation set.

Table 6.1: The mean errors over the last ten epochs over all folds for the five different parameter settings.

	Setting 1	**Setting 2**	**Setting 3**	**Setting 4**	**Setting 5**
Patch error	0.1307	0.2206	0.1319	0.1362	0.1280

After the training the mean validation error for the last ten epochs over all four folds was computed, separately for all five settings. These achieved errors can be seen in Table 6.1. The patch errors for setting 1, 3, 4 and 5 were similar, while the error for setting 2 was almost the double.

6.1.2 Images Run Through the Network

The classification network was trained on patches from benign images. Figure 6.3 shows an example of a whole benign image run through the final trained network. The image has not been used to produce patches for the training procedure. The output channels correspond to one class each and are thus the somewhat smaller segmentation images of the different parts.

Figure 6.4 shows an image of Gleason 5 tissue and the output when that was run through the same network. Even though the Gleason 5 image does not contain the same kind of tissue or parts as the benign images, there is tissue which resembles both

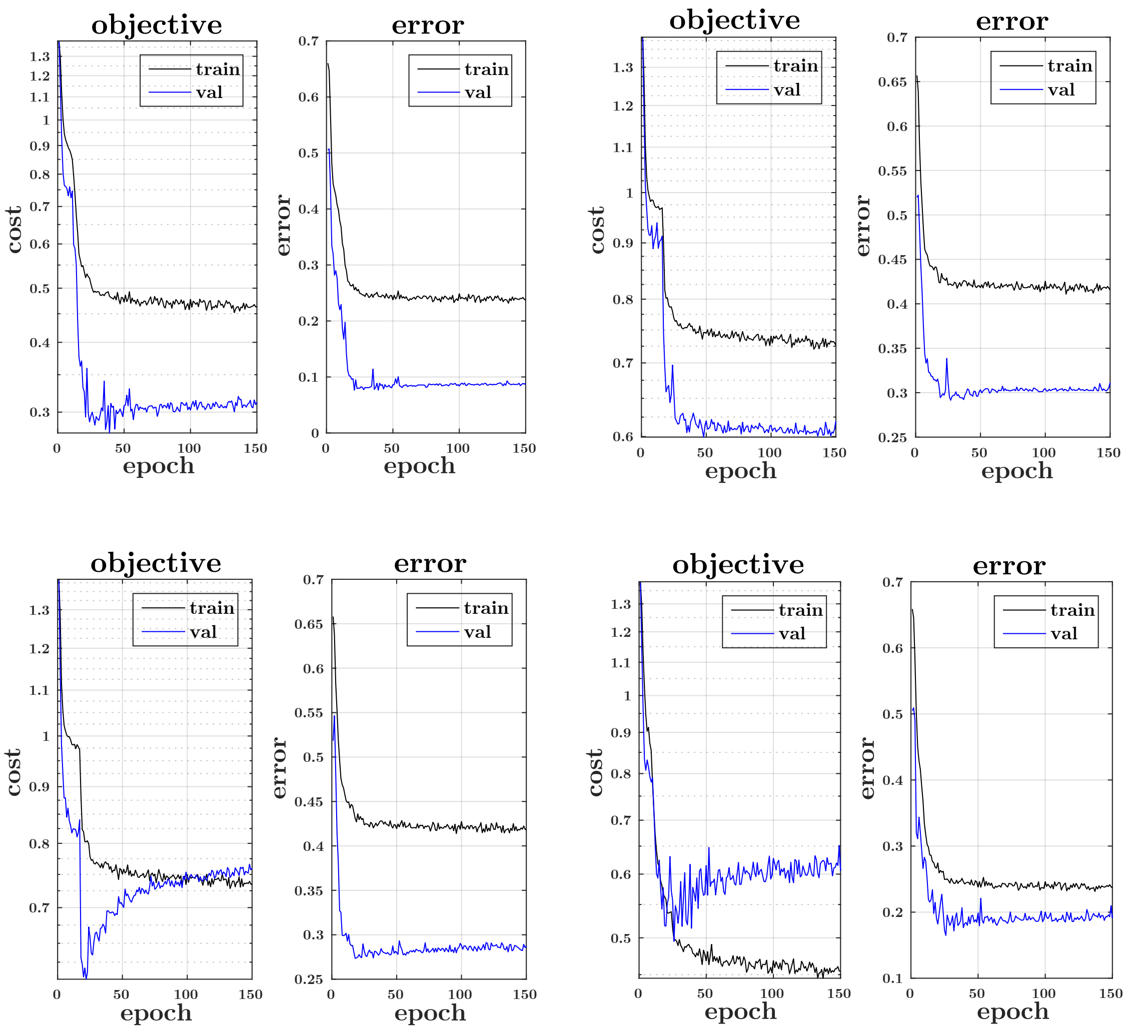

Figure 6.1: Objective and error plots for the segmentation network when using Setting 2 from Table 5.1. The four different objective-error-pairs are from the four different folds of the cross validation. The curves seem to have flattened after the 150 epochs, which indicates that the training is finished. Notice that the validation objectives and errors are lower than those for the training set due to dropout.

stroma and glandWallCells, while the outputs for glandWallPurple and whiteParts are very dark.

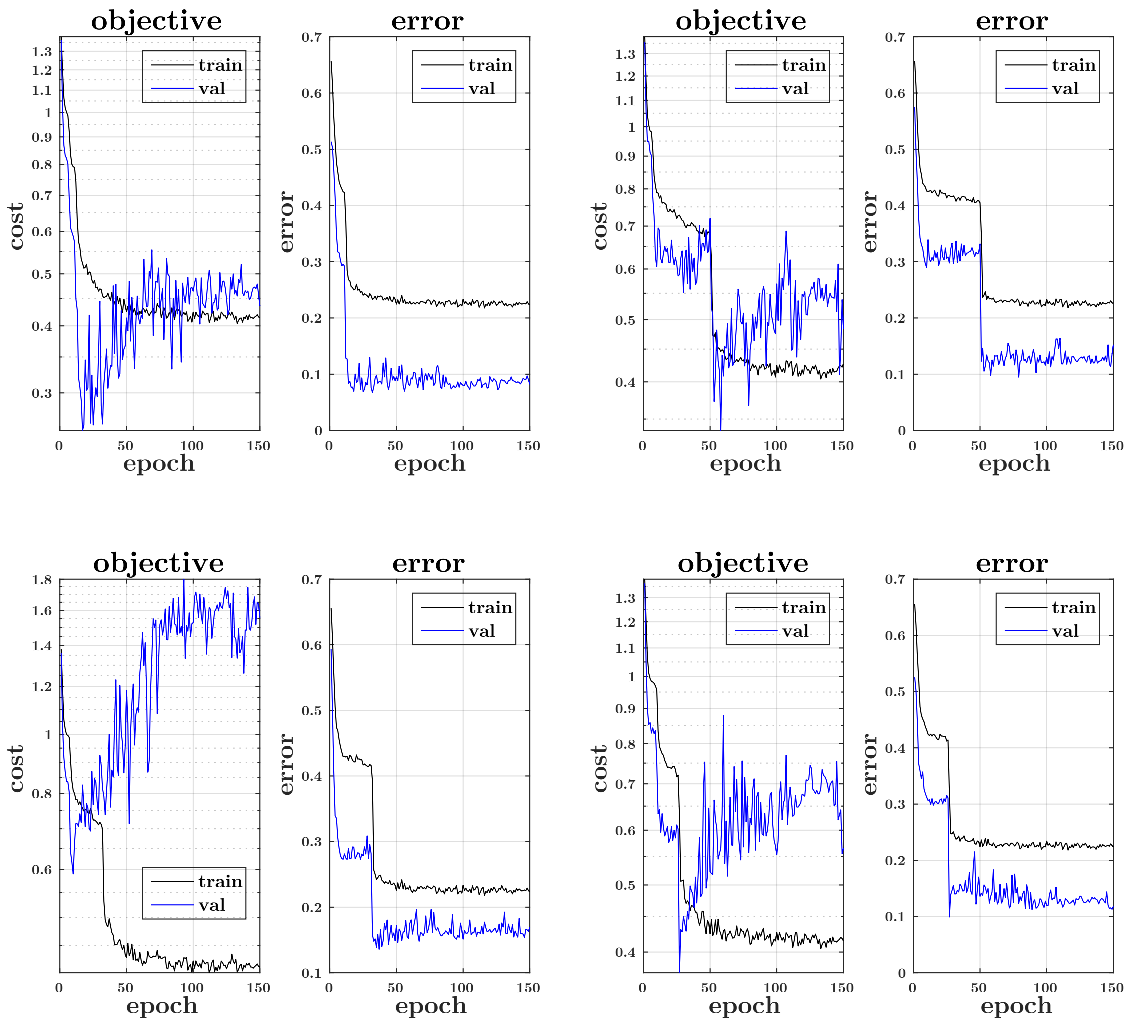

Figure 6.2: Objective and error plots for the segmentation network when using Setting 5 from Table 5.1. The four different objective-error-pairs are from the four different folds of the cross validation. The curves seem to have flattened after the 150 epochs, which indicates that the training is finished. In the graphs where the values for the validation set are lower than for the training set, this is due to the dropout.

6.2 The Classification Problem

For the classification network, three different approaches were taken. The objectives and errors that were achieved for these can be seen in Figure 6.5, 6.6 and 6.7. For the first and third approaches the training was run for 400 epochs, while it for the second approach was aborted after 90 epochs. The training is not done for any of the networks.

The mean errors — both patchError, imageError1 and imageError2 — for the ten last

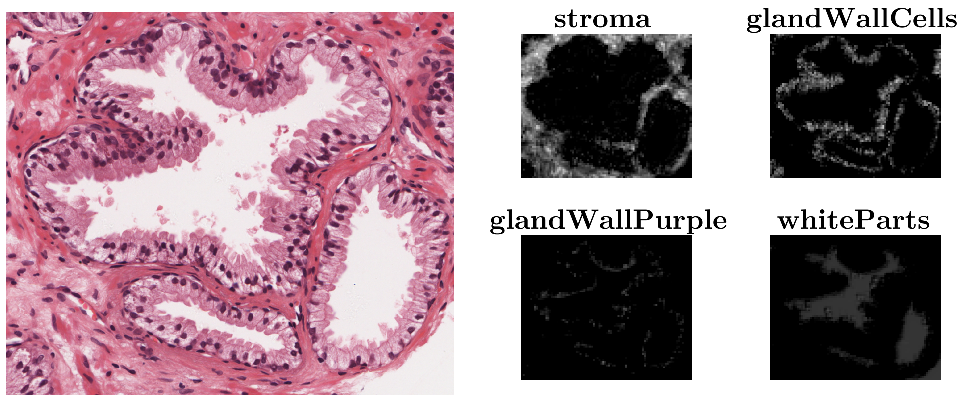

Figure 6.3: To the left, an image of benign tissue, to the right the output when that image was run through the segmentation network. The image shows the four different output channels. The segmentation seem to agree with the original image.

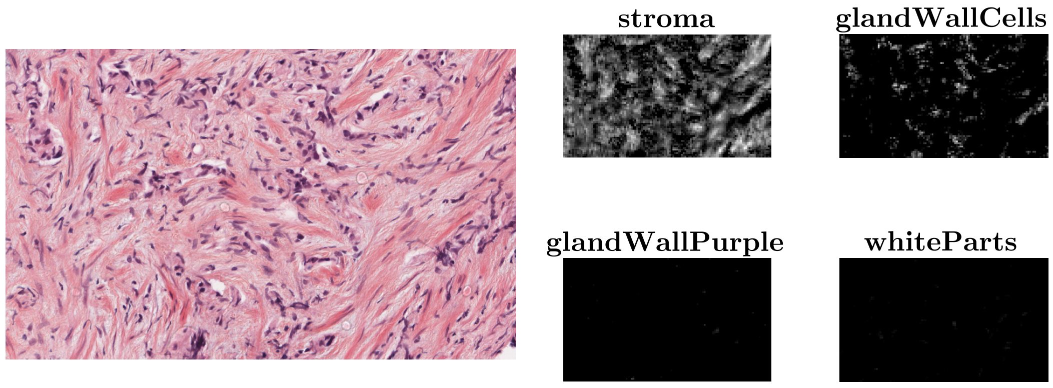

Figure 6.4: To the left, an image of Gleason 5 tissue, to the right the output when that image was run through the segmentation network. The image shows the four different output channels. The output indicates that the Gleason 5 tissue resembles both stroma and glandWallCells, but not glandWallPurple and whiteParts.

epochs can be seen in Table 6.2. The means are computed for the validation sets and only for approach 1 and 3, since the training for approach 2 was aborted earlier.

Table 6.2: The mean errors for the validation set over the ten last training epochs when the initialization of the parameters was done using approach 1 and approach 3. Approach 2 is left out since it was aborted.

	patchError	imageError1	imageError2
Approach 1	0.36	0.37	0.31
Approach 3	0.26	0.34	0.23

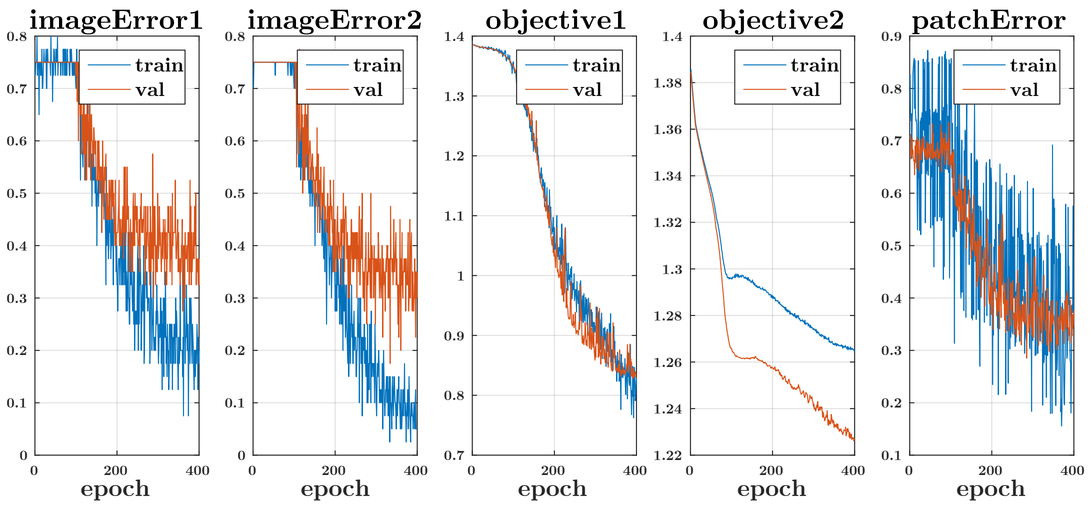

Figure 6.5: The errors and objectives that were achieved after 400 epochs for the first training approach of the classification network. The curves are still decreasing and this indicates that the training is not done.

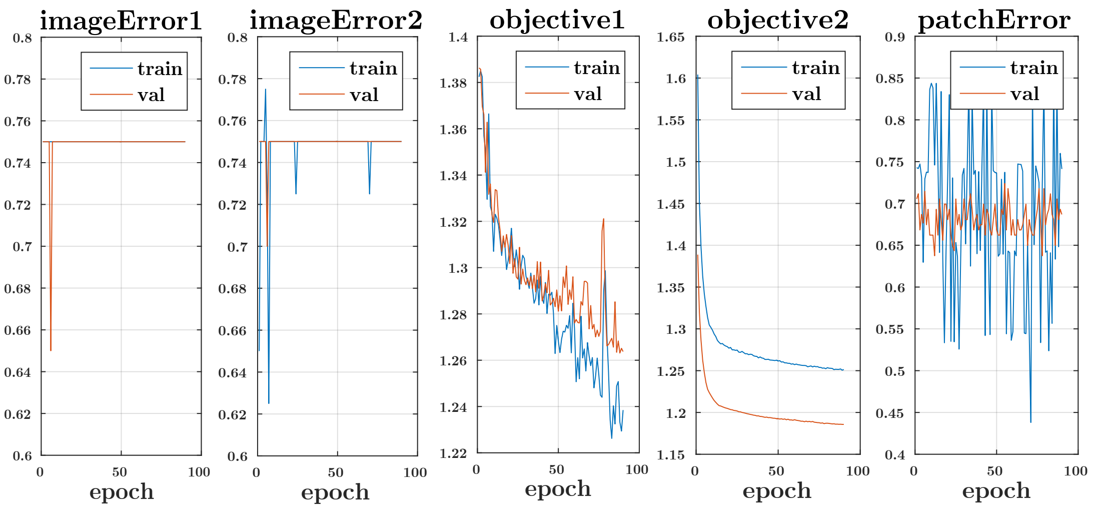

Figure 6.6: The errors and objectives that were achieved after 400 epochs for the second training approach of the classification network. No improvement had been achieved for the image errors when the training was aborted. Though, the decreasing trend of the objectives indicated that a continued training would give a better result.

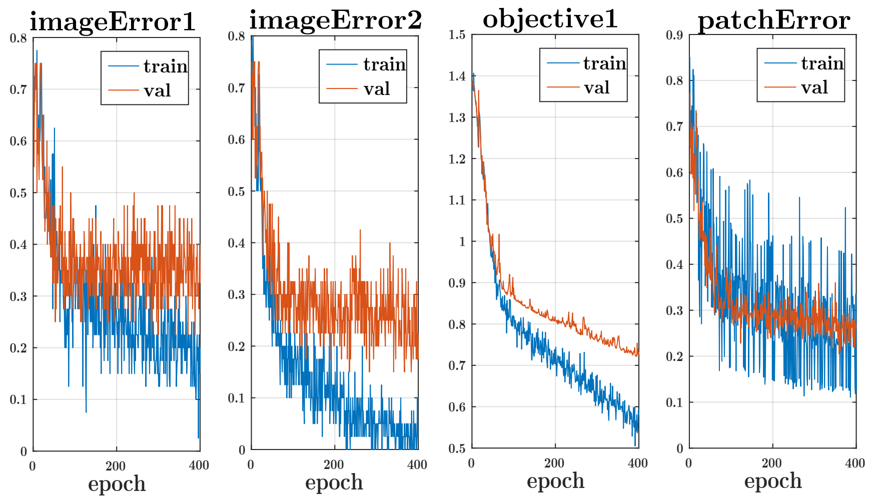

Figure 6.7: The errors and objectives that were achieved after 400 epochs for the third training approach of the classification network. In this figure, objective2 is not included, since it is constant throughout the training. The curves are still decreasing and this indicates that the training is not done.

7 Discussion and Further Work

One of the intermediate goals of this thesis was to find out whether the use of a middle key could be good for the final classification network. Since the segmentation network — without further fine tuning — did correct predictions for 87% of the input patches, it was decided that the contemplated approach could be useful for the final classification network.

Then, the main goal was to investigate the potential of different deep neural network structures that could be used for an automatic Gleason classification system, using this middle key. The final, trained networks have not achieved any remarkable results and they do not perform as good as the ones presented in Chapter 1.2. Nor are the networks as good as would be desirable for an actual usage at hospitals. Though, they do show potential. The training could be performed and without further tuning mediocre results were achieved. This thesis has thus fulfilled it's purpose as a pre-study of possible approaches to the problem. It has also contributed with innovation to the main DOGS project. Though, in the future there are many details that could be developed for better results.

7.1 The Dataset

To begin with, quite few images have been used for the training, especially for the classifying network. An expanded set of hand-made keys would have given more data to both networks, which could have given better results. Still, no more keys were produced due to the time it took to produce them. For the classification network the time consumption for the training was limiting, see further below. Since the thesis was mainly supposed to be investigating not too much focus was put on these parts.

Furthermore, as can be seen in Figure 6.4 it was not perfect to assign Gleason tissue the middle key class stroma everywhere, since there is tissue that resembles other tissue attributes. Actually, it is possible that the somewhat erroneous middle keys for the cancerous tissue counteracts the correct segmentation middle keys for the benign tissue. One future problem to attack could thus be to improve these, maybe by performing handmade segmentations of the cancerous tissue as well. Another approach could be to not assign the whole Gleason images the class benign, but a fifth class which is independent of the benign tissue. It could be called something like *cancerTissue*.

7.2 Implementation Issues

The implementations have been done using Matlab and they have been run on the computer CPU. The MatConvNet package has support for using the GPU. This could be one solution to the high time consumption, mentioned above. With a decreased time consumption, the number of images and input patches to the classification network could have been increased.

Another issue that arose was an out of memory-error when the classification network was run on more images (173 images). Each image yields a number of patches and each input patch is 316×316 pixels. This gives many weights, even if the layers are convolutional and not fully connected. The specific computer on which the computations were run of course have a large impact here. Another possible solution would be to down-sample the images before the patches are cut. Though, for the network to be built as has been explained above that would have had to be done before the segmentation network was created, which was not the case. The input patches to the segmentation network would also have to be smaller, since some tissue classes are quite narrow, as was explained in Chapter 5.1.1.

7.3 The Network Architecture

7.3.1 The Segmentation Network

A few different architectures for the simple segmentation network were tried, but finally it was set to the one presented in this report, without further investigation. First, the network was tried with only one fully connected layer, but the addition of another turned out to yield better results. The dropout does decrease overfitting, which was why it was added, see Figures 6.1 and 6.2. Though, a deeper or not as deep network could have made a difference. In the same manner, maybe a change in the number of filters could have made a difference. The activation function was set to always be ReLU and no others were tried.

7.3.2 The Classification Network

When it comes to the classification network, only the given architecture was tried. Since fully connected layers and dropout turned out to be successful for the segmentation network, it was added at the end of the classifying one as well. Though, it would have been interesting to see if it made any difference where the middle key was placed and how long the outgoing "branch" should be — that is how block5 in Figure 5.1 is built. Furthermore, it would be of interest to vary block3 and block4 in the same figure, to see what impact the number of layers have. By changing the number of layers in the network, the size of the input patches would automatically have to be changed as well.

These investigations have not been included in this thesis due to lack of time. The implementations of a DAG net have been performed and the architecture shows potential.

How large the potential is will not be established until the architecture is investigated further.

7.4 The Hyper-Parameters

The networks — both the one for the segmentation and the one for classification — have been run far more times than is shown in this report. Just to find hyper-parameter settings such that the objective functions do not explode and still change at all is a larger challenge than most reports about deep neural networks give the appearance of. Just small changes in one of the hyper-parameters might have a large impact on the final result.

For the segmentation a few different settings have been presented, see Table 5.1. The results turned out to be quite similar, except for when the learning rate η was changed, for setting 2. This probably does not have to do with the fact that it is adaptive, but with the size of it. Both if the learning rate is too high and too low, there is a risk of not reaching the objective minimum and as can be seen in Table 6.1 the difference in η in setting 1 and 2 made a large difference. Setting 5 turned out to produce the best results, but the differences to setting 1, 3 and 4 were small. Probably better results could have been achieved by tuning the hyper-parameters further.

For the classification network it turned out to be crucial with an adaptive learning rate. Otherwise, the objectives either exploded or did not change at all. Once a parameter setting which made possible for training was found, that setting was used, without further tuning. By using the same hyper-parameter settings for all three approaches, a more fair comparison of the approaches could have been made. This was not done since the larger batch size for approach 2 and 3 (Table 5.3 and 5.4) made the training more time consuming and a more effective setting worked for approach 1 (Table 5.2). If the work is continued, it would be a good idea to investigate whether the same settings would work for all different approaches.

Instead of having the adaptive learning rate as a step function, it could be implemented as a function of how the objective function has behaved for previous epochs. Moreover, the threshold δ for the middle key could have been found by investigating the output from the segmentation network. In general, to achieve the best possible results with a certain network architecture and a certain data set the tuning of the hyper-parameters should be done more methodically.

7.5 The Different Classification Network Approaches

To begin with, the predictions seem to be better according to imageError2 than to imageError1. Therefore, the second approach of computing the image prediction should be used. Thus, that corresponding error will also be the one referred to.

Three different approaches of initialization for the weights in the classification network have been tried. From the beginning the idea was to make a comparison between these, but since the networks are neither run until saturation, nor optimized a fair comparison cannot be made. For the same reason, there was no point in performing crossvalidation for these networks.

If any conclusions are to be drawn from the resulting plots of the three different networks, the third approach, where the parameters of block1, block2 and block5 are left untouched, shows the most potential. This is also the network that is mostly affected by the performance of the segmentation network. Though, as has been the case, this training was performed with a batch size $\tau = 100$, while the other two had a batch size $\tau = 200$. The larger batch size makes the training more time consuming and thus it was harder to change any of the other hyper-parameters at all.

It is regrettable that the training for the second approach — where the weights were initialized by the segmentation network but allowed to change — was not run until any improvements were achieved. Though, both objectives show a decreasing trend, so there is reason to believe that a decrease of the image error rate was to come, if the training would have been run further. Though, the combination of all parameters being trained and the large batch size made the training very time consuming — enough for it to be aborted.

If something should be said about the achieved error rates when using approach 1 and 3, the ones for approach 3 are lower after the 400 run epochs, imageError2 is 0.23 compared to 0.31 for approach 1. This is satisfying, since approach 3 uses the pre-training in the segmentation network more. Though, again, this would have to be further investigated to be ascertained.

8 Conclusion

The first network that was built was a segmentation network. This was a simple, straight network for classification of different tissue parts in benign prostate tissue. The classes were stroma, glandWallCells (nuclei of the cells making up the gland wall), glandWallPurple (the part of the gland wall cells that is not part of the previous class) and whiteParts (glandular lumen and background). The network classified 76×76 patches but did also perform segmentation of the mentioned tissue parts when it was applied to a larger image. The network classified 87% of the validation patches correctly and was also tried on larger images. The segmentation of a larger benign image was good. An image of Gleason grade 5 was also run through the network and the outputs showed tissue resemblance with stroma and glandWallCells.

Since the performance of the segmenation network was good it was decided that the idea with a directed acyclic graph network with a middle key consisting of a segmentation should be tried. There were keys for the benign images and the ones containing cancer were approximated such that their middle keys only contained stroma. The Gleason classification network was created such that the structure in the beginning was taken from the segmentation network and then additional layers were appended. Thus, this network was optimized using two objective functions.

The weights of the classification network were initialized in three different ways. In the first approach all weights were initialized as random. In the second approach the ones in the beginning were initialized to be the trained weights from the segmentation network, but they were allowed to be changed during training. In the third approach the weights were initialized in the same way as for the second approach, but the weights from the segmentation network were not allowed to change.

The training of the classification network turned out to be very time consuming. The second approach was because of this aborted after 90 training epochs but both objectives showed a decreasing trend when this was done. The other two approaches were run for 400 epochs but were then stopped, even though the training was not done. At this point the network from the first approach had achieved correct prediction of the images in 69% of the cases while the network from the third approach correctly classified 77% of the validation images.

The hyper-parameters were not optimized and the trainings were not run for perfection. This thesis has rather been about investigating the potential of a network with a middle key, for Gleason grade classification. The networks showed good possibilities to do correct classifications and achieved improving results even without optimization. Because of that it is concluded that the tried CNN architecture does have potential and that it would be a good idea to investigate and develop it further.

Bibliography

[1] Hanna Källén. *Applications of Machine Vision.* PhD thesis, Lund University, Faculty of Engineering.

[2] Wulfram Gerstner, Werner M Kistler, Richard Naud and Liam Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition.* Cambridge University Press, 2014.

[3] Jan-Erik Damber and Lars Grenabo. Urologi – Prostatacancer. In Jörgen Nordenström Bengt Jeppsson, Peter Naredi and Bo Risberg, editors, *Kirurgi.* Elanders Beijing Printing Co. Ltd.

[4] Cancerfonden. Cancerfondsrapporten 2016. `http://www.cancerfonden.se`, [2016-07-12].

[5] Vinnova. `http://www.vinnova.se`, [2016-07-12].

[6] Giuseppe Lippolis. *Image analysis of prostate cancer tissue biomarkers.* PhD thesis, Lund University, Faculty of Medicine.

[7] Hanna Källén, Jesper Molin, Anders Heyden, Claes Lundström and Kalle Åström. Towards grading gleason score using generically trained deep convolutional neural networks. In *2016 IEEE 13th International Symposium on Biomedical Imaging (ISBI)*, pages 1163–1167, April 2016.

[8] Geert Litjens, Clara I Sánchez, Nadya Timofeeva, Meyke Hermsen, Iris Nagtegaal, Iringo Kovacs, Christina Hulsbergen-van de Kaa, Peter Bult, Bram van Ginneken and Jeroen van der Laak. Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis. *Scientific reports*, 6, 2016.

[9] Anna Gummeson. Prostate Cancer Classification using Convolutional Neural Networks. Master's thesis, Lund University, Faculty of Engineering, 2016.

[10] Jonathan I Epstein, Michael J Zelefsky, Daniel D Sjoberg, Joel B Nelson, Lars Egevad, Cristina Magi-Galluzzi, Andrew J Vickers, Anil V Parwani, Victor E Reuter, Samson W Fine and others. A contemporary prostate cancer grading system: a validated alternative to the gleason score. *European urology*, 2015.

[11] Jonathan I Epstein. A new contemporary prostate cancer grading system. *Pathology international*, 2015.

[12] Andrew Ng et al. UFLDL Tutorial. `http://deeplearning.stanford.edu/tutorial/`, [2016-04-12].

[13] David Kriesel. A Brief Introduction to Neural Networks. available at `http://www.dkriesel.com`, 2007.

[14] Olivier Chapelle, Bernhard Schölkopf and Alexander Zien. *Semi-Supervised Learning.* The MIT Press.

[15] Andrea Vedaldi and Andrew Zisserman. VGG Convolutional Neural Networks Practical. `http://www.robots.ox.ac.uk/~vgg/practicals/cnn/index.html`, [2016-02-10].

[16] Michael A Nielsen. Neural Networks and Deep Learning. Determinaton Press, 2015, `http://neuralnetworksanddeeplearning.com`, [2016-02-10].

[17] Robert H Wurtz and Eric R Kandel. Central Visual Pathways. In James H. Schwartz Eric R. Kandel and Thomas M. Jessell, editors, *Principles of Neural Science.* McGraw-Hill Companies.

[18] Andrea Vedaldi and Karel Lenc. MatConvNet – Convolutional Neural Networks for MATLAB. In *Proceeding of the ACM Int. Conf. on Multimedia*, 2015.

[19] Regional Ethics committee at Lund University. `http://www.epn.se/lund/`, [2016-07-25].

Master's Theses in Mathematical Sciences 2016:E44
ISSN 1404-6342

LUTFMA-3304-2016

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/