Rotational Invariant Neural Networks for Prostate Cancer Classification

Joel Ekelund

Master's thesis
2017:E64

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Rotational Invariant Convolutional Neural Networks for Prostate Cancer Classification

Author:
Joel Ekelund
Supervisors:
Anders Heyden
Ida Arvidsson

Master's thesis

LUND
UNIVERSITY

Faculty of Engineering
Lund University
Centre for Mathematical Sciences
Mathematics

2017-09-29

Abstract

Prostate cancer was in 2012 the second most common type of cancer for males globally. To be able to treat prostate cancer in the most effective way it is important to know how aggressive the cancer is. This aggressiveness is graded using the Gleason score. The diagnosis is done by pathologists inspecting prostate biopsies, but the advancement of pattern recognition using Convolutional Neural Networks (CNN) has made it interesting to try to automate this process.
The data in this thesis is microscopy images of prostatic tissue. These images are rotation-invariant, meaning that they have the same Gleason grade no matter which angle they are inspected at. The goal of this thesis is to investigate the possibility of exploiting this rotation invariance to create a rotation invariant Convolutional Neural Network for automatic classification.
The rotation invariant CNNs include a rotation of filters, which makes interpolation an important aspect to investigate. This thesis does that by designing many different CNNs in a general way that have different sizes of the filters that are to be rotated. The resulting CNNs show that the filter size does indeed matter, with the smallest rotated filters that trained the network well were of size 17x17. The lowest resulting error rate for classification on both the training and validation data was 6.3%. The lowest error rate for classification on just the validation data was 16.7%, however the validation data consisted of only 24 images. The conclusion from this was that making the CNN rotation invariant can be of some interest, and could be investigated further by optimizing networks for a certain size of rotating filters.

Keywords: Convolutional Neural Networks, rotation invariance, deep learning, automated Gleason grading.

Acknowledgements

I want to thank my supervisors, Anders Heyden and Ida Arvidsson, for providing guidance and helping me focus the work to achieve the end-result. I also want to thank anyone with whom I have ever discussed the thesis.

Contents

1

Introduction

1.1 Background

Prostate cancer was in 2012 the fourth most common cancer in both sexes combined and the second most common type of cancer for males globally [1]. In Sweden in 2015 prostate cancer and breast cancer were the most common types of cancer [2].

To be able to treat prostate cancer in the most effective way it is important to know how aggressive the cancer is. The aggressiveness is graded using a score called the Gleason score. The grading is done by a pathologist who decides the score based on the appearance of a biopsy of prostatic tissue stained with for example hematoxylin and eosin. This can be fairly time consuming work, and in Sweden there is a lack of pathologists resulting in long waiting times [3]. There is also a possibilty of two different pathologists grading the same biopsy differently.

It is therefore interesting to try automating this process to classify the biopsies. An alternative for this automation is using Convolutional Neural Networks which can be trained to work for this specific type of task. The Network takes as input microscopy images of the stained biopsies and adjusts itself to become better at classifying the images. Moreover, the microscopic images are rotation invariant which often is used in Convolutional Neural Networks by rotating the images and using these rotated images as well as the original images to train the Network. However, another possibility when it comes to rotation invariant data is to make the Network rotation invariant in itself.

1.2 Problem formulation

The purpose of this thesis is to investigate the possibility of using rotation invariant Convolutional Neural Networks for classifying the prostate biopsies. A reason for using rotation invariant Networks would be to not need to expand the dataset by rotating the images used for training the Network. This would drastically reduce the amount of data the Network trains on which also reduces the time it takes for each training cycle. However, less data can also mean poorer results and overtraining of the Network.

The methods for making a Convolutional Neural Network rotation invariant are

not many. The method which is focused on in this thesis was first presented in [4], where it was used for texture classification. An issue that arises with making the rotation invariant layers is that of interpolation. In those layers, there are filters that are rotated 360° in steps, and when the angle is not a multiple of 90, interpolation is necessary. Therefore different sizes of filters will be investigated to see if there is a preferable size of filter.

1.2.1 Overview

The problem will be approached using Matlab and the toolbox MatConvNet. The toolbox does not include rotation invariant layers, so they will first have to be implemented - using the functions in MatConvNet.

After this some Network will be constructed that will be used to investigate how different sizes of filters in the rotating filterbank affects the results to see if the interpolation has a clear impact.

The report will first introduce the theory behind Neural Networks and especially Convolutional Neural Networks. Important information regarding the data will then be described. After these more theoretical parts, the method for completing the work on the thesis will be described, and then results will be presented and subsequently discussed.

2

Neural Networks

In this section the theory of Neural Networks (NN) will be described, starting at the simplest possible and working up towards a Convolutional Neural Network (CNN). The learning process and regularization techniques are also introduced.

2.1 Single Neuron

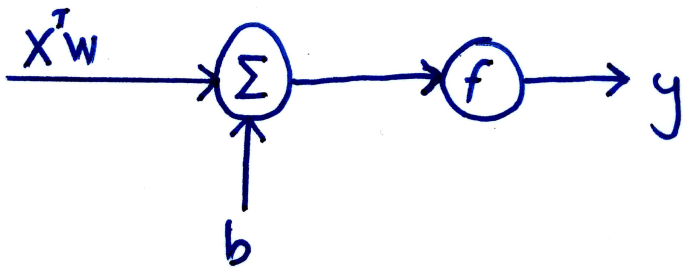

Figure 2.1: A Single Neuron

A single neuron would be the simplest possible Neural Network. A model of a single neuron can be seen in Figure 2.1. It takes an input vector x and has a weight vector w, a bias b and an activation function f, and creates an output y according to

$$y = f\big(\sum x_i w_i + b\big).$$

2.1.1 Activation Function

The most commonly used activation function in Convolutional Neural Networks is the Rectified Linear Unit (ReLU) which has been found to accelerate the learning of a Network several times[5]. The ReLU is defined as

$$f(x) = \max(0, x).$$

The ReLU simply thresholds the values at zero.

2.2 Multiple Neurons

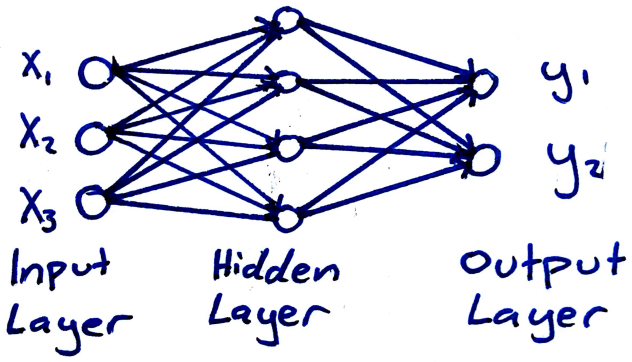

Figure 2.2: A Network consisting of an input, one hidden layer and an output.

A Neural Network usually consists of multiple neurons that are connected and organized in layers. Figure 2.2 shows a two layer feedforward NN with six neurons. Networks with many layers are sometimes called Deep Neural Networks, and the term deep learning is also sometimes used, since the networks get deeper when more layers are added. Feedforward means that the outputs from layer i are inputs to layer $i+1$, and there is no feedback. The Network has 26 learnable parameters, 20 weights and 6 biases. In regular Neural Networks it is common for the layers to be fully-connected. This means that each neuron in layer i is connected to each neuron in layer $i+1$, but neurons within a layer are not connected. In Convolutional Neural Networks it is common to end the Network with at least one fully-connected layer, but the earlier layers are usually not fully-connected. The output layer does not have an activation function, because it is for example in classification problems interpreted as the class scores. The output would for the Network in Figure 2.2 be

$$y_k = \sum_{j=1}^{4} w_{kj}^o f\big(\sum_{i=1}^{3} x_i w_{ji}^h + b_j^h\big) + b_k^o, \qquad k = 1, 2,$$

where the superscript h is for the hidden layer, and o is for the output layer, and the subscripts describe the weights connecting the different layers, for example w_{21}^o is the weight connecting the upper neuron in the hidden layer with y_2.

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are similar to regular Neural Networks in that they consist of neurons with learnable weights and biases. The difference is that with CNNs the input is usually images. Earlier the inputs have been 1D-vectors, but for a CNN the inputs are 2D (grayscale) or 3D (color) volumes. Regular Neural Networks do not scale well to images. The image-sizes used for learning in this project were 128x128, which gives a single fully-connected neuron in the first hidden layer $128 \cdot 128 = 16384$ weights to learn.

The CNN takes advantage of knowing the inputs will be images so the layers have the neurons arranged in three dimensions (width, height and depth). The layers will be organized so that they rescale the input so that the ouput will be a 1x1xC volume, where C is the number of classes in the classification problem.

2.3.1 Typical Layers

A CNN typically consists of a few different types of layers, with the convolutional layer the only one that is needed for the Network to be called a Convolutional Neural Network. Since the inputs to a CNN are images, the neurons are here spatial filters.

2.3.1.1 Convolutional Layer

The convolutional layer is the core layer of a CNN. It takes as input a 3D volume and produces a 3D volume output. For example if you have RGB-images, the input volume will be width x height x 3. The depth of the output volume depends on the number of filters in the filter bank.

The operation performed in the convolutional layer is a convolution, which when performed between the data in x with a kernel w is defined as

$$(x * w)(s) = \int_{-\infty}^{\infty} x(u)w(s-u)du,$$

or discretely as

$$(x * w)(s) = \sum_{u=-\infty}^{\infty} x(u)w(s-u).$$

The discrete function can then be expanded to two dimensions as

$$(x * w)(s,t) = \sum_{u=-\infty}^{\infty}\sum_{v=-\infty}^{\infty} x(u,v)w(s-u,t-v).$$

The two dimensional discrete convolution operation is the relevant one when convolving images with kernels. The interpretation of this is that the kernel is flipped both vertically and horizontally and then slid over the image and the Frobenius inner product is calculated on the overlap.

Example A part of an image has the values $\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$, and the kernel is $\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$. The flipped kernel will then be $\begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}$, and the result from the Frobenius inner product between input and kernel is: $-1 \cdot 1 + 0 \cdot 2 + 0 \cdot 3 + 1 \cdot 4 = 3$. □

The convolution operation can change the size of the input volume, depending on the kernel-size, number of filters, stride and zero-padding. The stride describes how much the kernel is moved in each step. With a stride of one pixel the kernel is moved one pixel at a time, with a stride of two pixels the kernel is moved two pixels at a

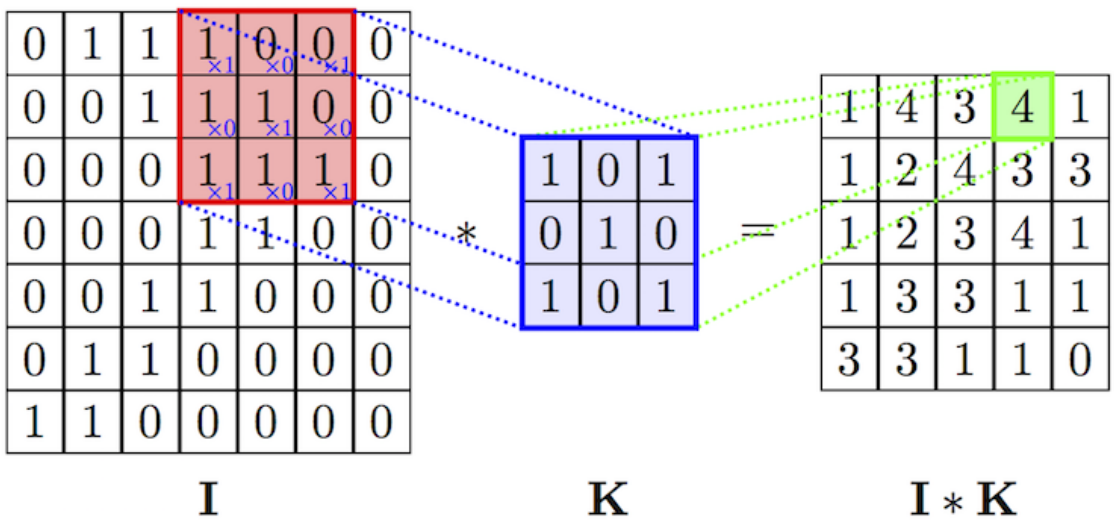

Figure 2.3: This image shows the convolution between an image I and a kernel K. It highlights one step in the convolution, [6].

time, and so on. A larger stride will reduce the width and height of the output more. Zero-padding can be used to counter this reduction of size. It is just the operation of padding the input with zeros around the border. For example, an input that is convolved with a kernel of size 3x3 with stride 1 and no zero-padding will have its width and height reduced by 2 pixels. The kernel size, stride and zero-padding changes the width and height of the input volume while the number of filters will change the depth.

Figure 2.3 shows an example of a 7x7 matrix being convolved with a kernel of size 3x3, producing a 5x5 output matrix. The highlighted parts show how the green 4 was achieved by convolving the red area with the kernel. The rest of the output comes from the other possible overlaps between the matrix I and the kernel.

2.3.1.2 Pooling Layer

Pooling layers are used to reduce the spatial size to reduce the number of parameters. The pooling operation works independently on each depth slice. The most common type of pooling is max-pooling. It is common to use 2x2 pooling filters with stride 2, which will basically divide the input into 2x2 squares and the output will be the maximum values for all of these squares. This will reduce the number of activations by 75%. Figure 2.4 shows the max-pooling operation. There are other types of pooling, for example average-pooling, which instead would take the average of the values in the 2x2 squares.

2.3.1.3 Softmax-function

It is common to end the CNN with a softmax function that will make the results easier to interpret in a classification problem. The softmax function is defined as

$$f_j = \frac{e^{z_j}}{\sum_k e^{z_k}},$$

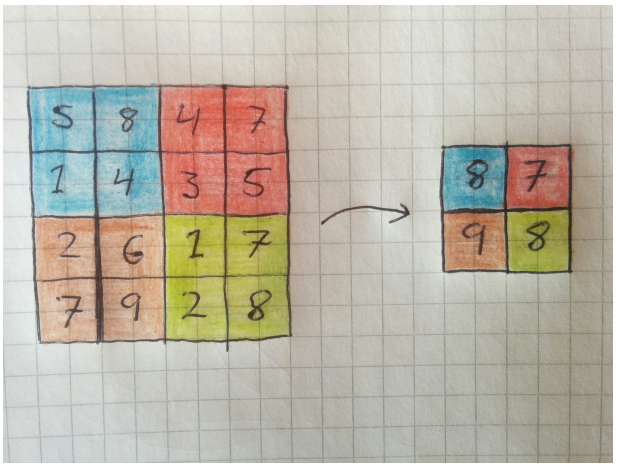

Figure 2.4: Maxpooling with a 2x2 pooling filter with stride 2.

where z is a vector of arbitrary scores, j is the class and the sum in the denominator is over all classes. The softmax function takes the scores in z and converts this to a vector of values, f, where the elements have values between zero and one and the sum of all elements in the vector is one. These values can then be interpreted as probabilities.

2.3.2 Rotation-Invariant Layers

The layers described in Section 2.3.1 are the traditionally used layers for Convolutional Neural Networks. These are not rotation invariant by themselves so there is a need for adding some other layers to make the Network rotation invariant. There are two layers that are used in this project to make the Network rotation invariant. One is a rotation of the filters in the filterbank, and the other is an orientation max-pool operation.

2.3.2.1 Rotating Filterbanks

This layer is an extension of a regular convolutional layer in the Network. The filterbank is extended so that for each filter in the filterbank, 360° rotations of the filter in a certain amount of steps is added to the filterbank. Or, if the first filter in rotation group i is denoted h_0^i, then after this filter the filters

$$h_\alpha^i = \text{rotate}(h_0^i, \alpha), \qquad \forall \alpha \in [1, \ldots, R]\frac{360}{R},$$

where R is the number of steps in the 360° rotation, are added in connection with the first filter, h_0^i. When rotating the square filters by an angle that is not a multiple of 90°, parts of the rotated filter would be outside of the filter. This is fixed by only using pixels in a circle with center at the center of the filter and a diameter the width of the filter. For those rotation angles interpolation will also affect the rotated filters.

2.3.2.2 Orientation Max-pooling

The orientation max-pooling operation is similar to the regular max-pooling operation, but works on the whole depth-slices from one rotation group. Instead of taking

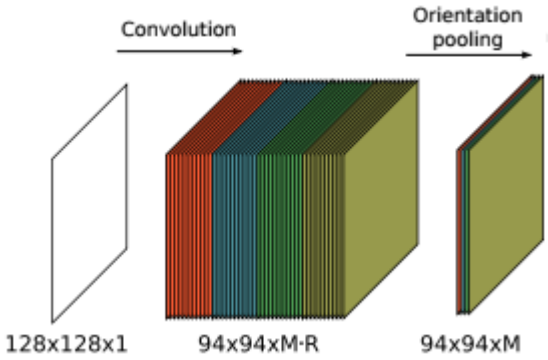

Figure 2.5: The two layers used to make a Network rotation invariant [4].

the max value from a certain area of one depth-slice as in regular max-pooling, it sends through the whole depth-slice that has the largest activation in one rotation group.

Figure 2.5 shows an illustration of how both the rotating filterbank and the orientation max-pooling works. In the illustration the input is a grayscale image of size 128x128, and the filters in the convolution are 35x35. To the left in the image is the input. In the middle is the result from the convolution with the extended filterbank, where each color represents one rotation-group. To the right is the result after the orientation max-pooling, where one depth-slice from each rotation group remains.

2.4 Training

Training a CNN is the process of using as input to the CNN images with a known class, and using the comparison of the classification result from the CNN with the real class to update the weights of the Network.

2.4.1 Objective Function

To compare the classification result with the real class, an objective function is needed (sometimes called cost function or loss function). The objective function should take the classification result and the real class and use this to produce a high number if the classification result is bad, and a low number if the classification result is good.

A common objective function is the softmax log loss, which is a combination of the softmax-function from Section 2.3.1.3 and multinomial logistic loss.

$$L_i = -\log \text{softmax}(z)_i = -z_i + log \sum_j e^{z_j}.$$

This is the loss for an image, and the total loss is the average of this for all data

$$L = \frac{1}{N} \sum_i L_i,$$

where N is the total amount of images.

2.4.2 Backpropagation

With the loss function defined, what is now needed is the gradient of the loss function with respect to all the different weights in the Network. The gradient is needed for the optimization method Stochastic Gradient Descent with Momentum, which is described in Section 2.4.3.
The gradient is propagated backwards through the Network, starting from the gradient of the loss function, by applying the chain rule until the wanted gradient is found.
To backpropagate through a CNN, one needs to know how the backpropagation through the specific layers work. Backpropagation through a convolutional layer is a convolution but with the filters rotated 180°. Backpropagation through a max-pooling layer is passing the gradient to where it came from (back to the coordinate that won in the max-pooling). Backpropagation through an average-pooling layer is dividing the gradient equally over the whole pooling block. Backpropagation through the orientation max-pooling is just going back through the activated filter (rotated 180° as it is a convolution) in the rotation groups.

2.4.3 Stochastic Gradient Descent with Momentum

To minimize the loss function the algorithm Stochastic Gradient Descent with Momentum is used. It can be broken down into smaller parts when described. Stochastic gradient descent is also sometimes called online gradient descent.[7, 8]

2.4.3.1 Gradient Descent

Gradient Descent is a minimization procedure where the weights, w, are updated iteratively according to

$$w_{t+1} = w_t - \Delta w_t, \text{ where } \Delta w_t = \eta \frac{\partial L}{\partial w},$$

where η is the learning rate, which is an important parameter to tune when training the Network and the subscript indicates a step in the process.

2.4.3.2 Batch Gradient Descent

Batch Gradient Descent is an extension of regular Gradient Descent in that a batch of m examples is used for the update instead of all examples.

$$w_{t+1} = w_t - \eta \frac{1}{m} \sum_{i=1}^{m} \frac{\partial L_i}{\partial w}.$$

2.4.3.3 Stochastic Gradient Descent

With Stochastic Gradient Descent (SGD), a single example (k), or a subset of examples, is chosen at random from a batch in the training set at each iteration and an approximation of the true gradient is then used to update the weights according to

$$w_{t+1} = w_t - \eta \frac{\partial L_k}{\partial w}.$$

2.4.3.4 Stochastic Gradient Descent with Momentum

Extending SGD to include momentum means including another term in the update that remembers the last update, as

$$\Delta w_{t+1} = \eta \frac{\partial L_k}{\partial w} + \alpha \Delta w_t.$$

The momentum term α determines the strength of the previous gradient.

2.4.4 Regularization

When training a Convolutional Neural Network, a problem that might occur is overfitting, which is when the Network learns details and noise in the training images in addition to the details that define the images. This problem can be called generalization error, which refers to how well the learned Network works for examples not included in the training. Overfitting can be observed by looking at how the errors evolve over training epochs - if the training error is much lower than the validation error, which might even start to increase - then the Network is probably overfitting to the training data. Regularization is the name for methods which are supposed to reduce the overfitting, or as Ian Goodfellow defines it in Deep Learning *"any modification we make to the learning algorithm that is intended to reduce the generalization error, but not its training error"*[9]. The regularization methods that are used in this project are described below.

2.4.4.1 Weight Decay

Weight decay is a regularization strategy that adds the term $\frac{\lambda}{2}\|w\|_2^2$ to the objective function, where λ is the term that is changed when weight decay is said to be changed. This will drive the weights closer to the origin and prevent overfitting. Weight decay is also known as the L^2 parameter norm penalty or Tikhonov regularization.

2.4.4.2 Data Augmentation

A good way to improve the generalization of the Network is to train it on more data. However, the amount of data is often limited and acquiring more data can be hard or impossible. One way to get around this is to expand the dataset artificially. Some methods to do this is to add noise to the data, enlarge or shrink the data, rotate the data, translate the data or flip the data and then adding this new data to the dataset. All methods can not be used for all types of data. For example, rotating or flipping the data is not a good method when dealing with classification of characters, since if you rotate a 'd' 180° it becomes a 'p'.

2.4.4.3 Dropout

Dropout is a technique that drops out neurons in a Neural Network, which means that it temporarily removes the neuron from the Network. The principle is shown in Figure 2.6, where to the left is a Network without dropout with full connections between the layers, and to the right is the same Network with dropout which is much more sparsely connected. Which neurons that are dropped is random and a

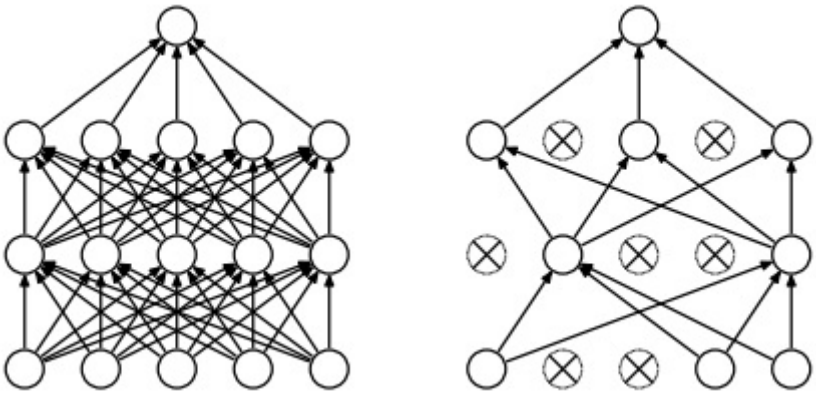

Figure 2.6: To the left is a Network with two hidden layers with full connections. To the right is the same Network with dropout applied, [10].

neuron has a probability, p, to be dropped and is often set around 0.5. The forward propagation, backpropagation and learning is performed as usual. Dropout is only applied during the training.

A reason for the improvements when using dropout is that the neurons in the hidden layers need to learn useful features on their own without relying on other neurons to help them. Dropout can also be seen as an efficient way of performing model averaging. Averaging the predictions from several different Networks is a good way to reduce errors, but the training and testing of several different Networks is computationally expensive. Dropout makes it possible to train a large amount of different Networks in a reasonable time [11].

3

Dataset

3.1 Images

The data set consists of 213 microscopy images taken with 40x magnification of sliced biopsies of prostatic tissue that have been stained with hematoxylin and eosin. The images differ slightly in colour possibly between the two sources they come from, the Prostate Cancer Research Consortium and PathXL in Belfast and Beamount Hospital in Dublin.

The images are divided into four different classes according to their Gleason grade. There are 52 images in the benign class, 52 images graded with Gleason grade 3, 52 images graded with Gleason grade 4 and 57 images graded with Gleason grade 5. Figure 3.1 shows an example image for each of the four different classes. Eight of the images were however too small to use.

3.2 Gleason score

The Gleason score was first devised by Donald Gleason in the 1960's. The score is based on the patterns in the microscopy images of the prostate biopsy. The biopsies are graded on a scale from 1 to 5, where grade 1 means that the biopsy looks almost like healthy prostate tissue and grade 5 means that the cells have mutated to the point of barely resembling normal prostatic tissue. The full Gleason score is derived by finding the most prevalent pattern and the second most prevalent pattern and adding the grades of these together to get a Gleason score between 2-10. The lower scored cancers are less aggressive than the higher scored. Lately the lowest grade that is given is grade 3. [12]

Since the introduction of the Gleason score in the 1960's there have been some changes to the classification. Recently a total scale from 1-5 has been suggested. This means that there are now only five different total scores instead of 25 (1+1, 1+2, 2+1,...) [13], however as mentioned, the grades 1 and 2 are not assigned anymore. The data used in the thesis are images with one Gleason grade each: benign, grade 3, grade 4 and grade 5. The images are a part of a larger full microscopy image. So the small images used in this thesis are graded and then they can be looked at in the larger scale to see which grades are the most common and second most common for the full microscopy image to give the final score. This final score

is the one that recently has been suggested to be between 1-5. The description of these final scores and the grade combinations that these scores represent can be seen in Table 3.1.

Table 3.1: Description of the full Gleason scores [13].

Full Score	Description
1 (Gleason grade 3+3=6)	*Only individual discrete well-formed glands*
2 (Gleason grade 3+4=7)	*Predominantly well-formed glands with lesser component of poorly formed/fused/cribriform glands.*
3 (Gleason grade 4+3=7)	*Predominantly poorly formed glands with lesser component of well-formed glands.*
4 (Gleason grade 8)	• *Only poorly formed glands, or* • *Predominantly well-formed glands and lesser component lacking glands, or* • *Predominantly lacking glands and lesser component of well-formed glands*
5 (Gleason grades 9&10)	*Lack of gland formation with or without poorly formed glands.*

3.3 Data Augmentation

The microscopic images of prostatic tissue are both rotation invariant and flip invariant. So the data set is extended by flipping the images up/down and left/right. It is also possible to rotate the images and add these to the data set, however this is not wanted when looking at the possibility of a rotation invariant network since a point of the rotation invariant network is to not have to expand the data set in this way to reduce the number of training images.

The images in the data set come in different sizes, ranging from 491 568 pixels to 40 086 725 pixels. A CNN requires the images used to be the same size, therefore patches of size 128x128 pixels where extracted from the images after the images were resized by a factor 0.15. The patches were extracted from the regular images and the images that were flipped up/down and left/right. The number of patches extracted from each image depended on the size of the image. From some images no patches were extracted due to the images being to small. The images were also normalized to values between 0-255 for each color channel independently.

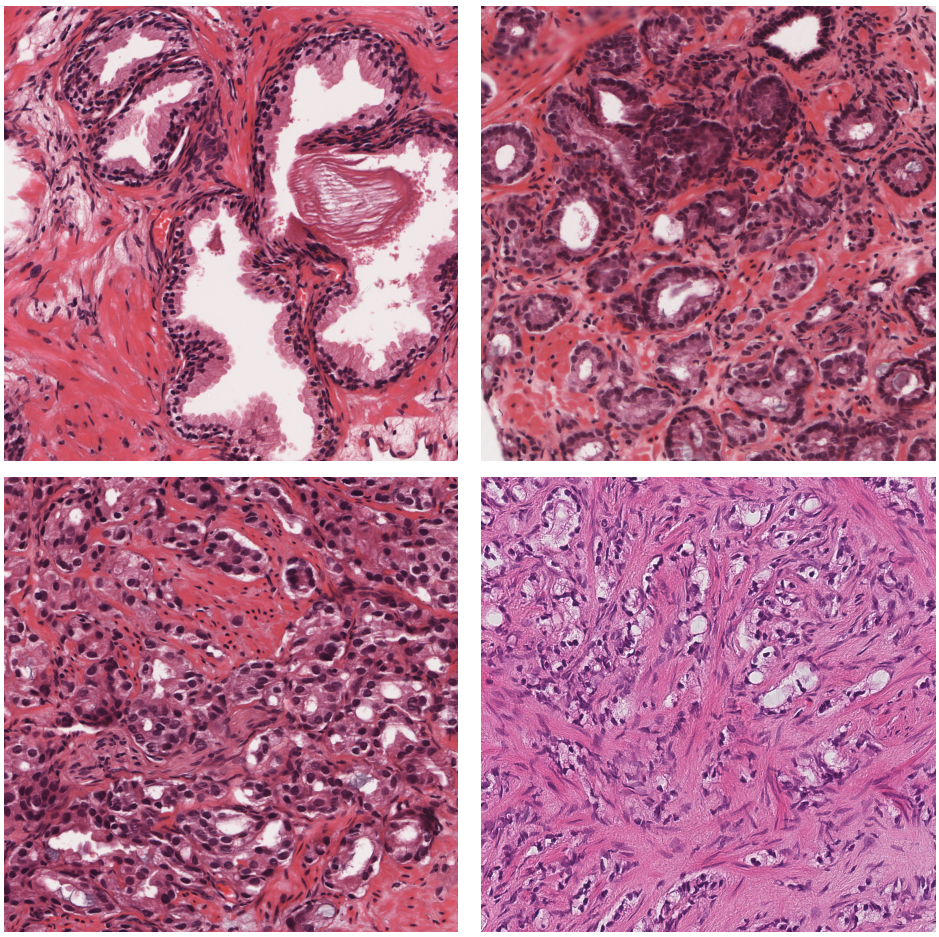

Figure 3.1: Microscopic images from the four different classes used in this project. Upper left: benign, upper right: Gleason grade 3, lower left: Gleason grade 4, lower right: Gleason grade 5.
The difference in colour between the image sources is visible between the grade 5 image and the rest.

4

Method

4.1 Software

The programming in this thesis was all done using Matlab, and the toolbox MatConvNet which implements Convolutional Neural Networks. More information on MatConvNet can be found at the projects website, `http://www.vlfeat.org/matconvnet/`.

4.2 Implementation

Implementing a regular Convolutional Neural Network using MatConvNet is mostly a matter of designing the Network using the already implemented layers in the toolbox. The important layers such as the convolutional layer, pooling layers, activation functions and loss functions are directly available through the toolbox. The rotation invariant layers described in Section 2.3.2 must be implemented. In MatConvNet what is needed for this is implementing a forward function for the forward propagation and a backwards function for the backpropagation. The learning process using Stochastic Gradient Descent with Momentum is also already implemented in the toolbox.

4.3 Process

With all the layers implemented and available, the next step is to find viable CNNs for the problem at hand. This means trying different strutures of the Network and many different hyperparameter combinations. This is a very time consuming work since the learning process takes some time, and since there can be so many combinations that performs similarly. Changes can be made to the learning rate, weight decay, filter size and stride, zero-padding, depth in each convolutional layer, pooling sizes, dropout rate etc.

When some Network structure and certain hyperparameter-combinations were found that worked somewhat well for different sizes of the rotational filters (since that is something that is investigated in the thesis), a problem that arises is how to make it a fair comparison between different sizes. This is a problem because of how the convolutions change the size of the output depending on the size of the filter. In the

end the method used for a somewhat fair comparison is using different amounts of zero-padding for the different filter-sizes so that the output of the rotational layers will always have the same size. Table 4.1 shows the design of the Network used for testing different sizes of rotating filters. The structure used is so that the convolutional layers do not change the height and width of the input, except the fully connected convolutional layer that has size 4x4. The convolutional layers with size 3x3 have a stride of 1 and a zero-padding of 1. The data size column shows the size after the data has gone through the current layer.

Table 4.1: The structure of the Network used to try different sizes for the rotational filters.

Type	Size	Depth	Data size
Input			128x128x3
Rotation Invariant	Variable	16	128x128x16
Maxpool	2x2, Stride 2		64x64x16
Conv	3x3	50	64x64x50
Maxpool	2x2, Stride 2		32x32x50
Conv	3x3	100	32x32x100
Maxpool	2x2, Stride 2		16x16x100
Conv	3x3	150	16x16x150
Maxpool	2x2, Stride 2		8x8x150
Conv	3x3	50	8x8x50
Maxpool	2x2, Stride 2		4x4x50
Dropout	Dropout rate 0.5		4x4x50
Conv	4x4	16	1x1x16
Dropout	Dropout rate 0.2		1x1x16
Conv	1x1	4	1x1x4

The variable sizes used were from 3x3 up to 35x35 in increments of 2. For all of these sizes the different combinations of learning rate and weight decay that is shown in Table 4.2 were used, divided into groups. Since the tests on different sizes of rotational filters were done for sizes from 3x3 to 35x35 in increments of 2 for six different combinations of learning rate and weight decay the resulting amount of Networks are $17 \cdot 6 = 102$. To train all these Networks on the full dataset would take too much time, so these were trained on a subset of the dataset, with 200 subimages from each class to train on. The Networks that produced somewhat promising results from this training were then trained on the full dataset.

All of this above will give a Network that classifies the small patches that were extracted from the images. However, when diagnosing, the classification of a full

Table 4.2: The different combinations of learning rate and weight decay used, divided into groups.

Learning Rate	Weight Decay	Group
10^{-3}	0	1
10^{-3}	0.1	2
10^{-4}	0	3
10^{-4}	0.1	4
10^{-5}	0	5
10^{-5}	0.1	6

image is wanted. This is created by dividing the image in patches as before, but not flipping anything. These patches will be classified as usual, and then the most common classification on the patches from one image will be the classification of the full image. Meaning that if nine patches is extracted from one image and six of them are classified as benign and the other three are classified as grade 3 then the full image will get the classification benign. This will be done for all images, both the ones included in the training set and in the validation set. It is done on both the training and validation data to see how these classifications might differ in accuracy. The validation data only includes six images per class (this is around 10% of the total amount of data), which means 24 images in total, which is not a large enough amount to draw any conclusions from the resulting error rates, but they will still be included. Ideally, there would be another separate test-set to do this on, but the amount of data used was too small.

5

Results

5.1 Classification on Patches

From the first tests, using the smaller subset of training images, on all of the different filter sizes with the six different combinations of learning rate and weight decay there were 19 results that were deemed promising enough to train on the full dataset. These can be seen in Table 5.1, which shows the filter size, learning rate and weight decay and also the group the Networks were designated based on their learning rate and weight decay.

Table 5.1: The parameters for the Networks that showed promising results on the smaller dataset divided into groups corresponding to their learning rate and weight decay.

Filter Size	Learning Rate	Weight Decay	Group
3x3	10^{-3}	0	1
5x5	10^{-3}	0	1
17x17	10^{-3}	0.1	2
19x19	10^{-3}	0.1	2
23x23	10^{-3}	0.1	2
25x25	10^{-3}	0.1	2
27x27	10^{-3}	0.1	2
17x17-35x35	10^{-4}	0	3
29x29	10^{-4}	0.1	4
31x31	10^{-4}	0.1	4

In Table 5.1 there are only two Networks with filters that are small, 3x3 and 5x5 both from parameter-group 1. The Networks trained with the parameters in group 5 and 6 did not give any promising results. Figure 5.1 shows the resulting objective function and error over 50 epochs for six of these 19 Networks, when they were trained on the smaller dataset. The error is in the Figure called top1err, simply because the

classification ranks the scores for the different classes and top1err means that the largest scoring class is either correct or false.
The 19 Networks that were trained on the full dataset showed similar result with each other within the same groups of learning rate and weight decay. Figure 5.2 shows a representative example of the objective and error for each group.
Figure 5.3 shows how the objective and error evolves for two different examples that were trained with group 3 parameters for 50 epochs and then with an added weight decay of 0.1 for 25 epochs. This was done as an attempt to reduce the overfitting.

5.2 Classification on Full Images

For the classification on the full images the results will be presented as an error rate for the Networks. There will also be confusion matrices for some of the Networks to see how some of the mis-classified images were classified. The error rates and confusion matrices will be based on the final epochs of the different Networks, some of them trained for 50 epochs and some for 75. The Networks used for this were the Networks mentioned in Table 5.1. The results will be presented for each group separately.

5.2.1 On Training and Validation Data

In this section the results are from the classification on both the training and the validation images.

Group 1 The Networks in group one were the only ones with small filters. This proved to be a poor Network to use, and the resulting error rate was for the Network with filter size 5x5 around 75%. Looking at the confusion matrix it becomes clear that the Network has simply classified all the images as benign, see below. In the confusion matrix the rows represent the actual class and the columns represent the predicted class.

$$\begin{bmatrix} 51 & 0 & 0 & 0 \\ 51 & 0 & 0 & 0 \\ 49 & 0 & 0 & 0 \\ 54 & 0 & 0 & 0 \end{bmatrix}$$

Group 2 For group 2 there were five different filter sizes that were trained further with the full dataset, see Table 5.1. The resulting error rates for the different filter sizes can be seen in Table 5.2. The error rates are spread from 18.0% up to 33.7%. Below are the confusion matrices for the best (19x19, left) and the worst (25x25, right) Networks.

$$\begin{bmatrix} 51 & 0 & 0 & 0 \\ 6 & 26 & 13 & 6 \\ 0 & 6 & 39 & 4 \\ 0 & 2 & 0 & 52 \end{bmatrix} \quad \begin{bmatrix} 51 & 0 & 0 & 0 \\ 20 & 25 & 2 & 4 \\ 1 & 28 & 20 & 0 \\ 0 & 12 & 2 & 40 \end{bmatrix}$$

For both those Networks, all of the benign images were correctly classified. However, the poor Network seemed to classify more to the lower grades, as 20 of the grade 3 images were classified as benign and 28 of the grade 4 images were classified as grade 3.

Table 5.2: Error rates for the Networks in group 2.

Filter size	Error rate
17x17	21.0%
19x19	18.0%
23x23	18.5%
25x25	33.7%
27x27	26.3%

Group 3 Group 3 had 10 different filter sizes that were trained on the full dataset. Out of these, 7 were trained a further 25 epochs, with an added weight decay of 0.1, compared to the rest in an attempt to reduce overfitting. Table 5.3 shows the resulting error rates for the different filter sizes, the ones trained for 25 epochs more are in bold.

Table 5.3: Error rates for the Networks in group 3.

Filter size	Error rate
17x17	6.3%
19x19	7.3%
21x21	9.8%
23x23	8.3%
25x25	8.3%
27x27	11.2%
29x29	7.3%
31x31	18.5%
33x33	8.3%
35x35	9.3%

The results in Table 5.3 shows that the Networks trained for another 25 epochs had lower error rates. Below are the confusion matrices for the Networks with filter sizes 17x17, 29x29, 31x31 and 35x35, in that order.

$$\begin{bmatrix} 50 & 1 & 0 & 0 \\ 1 & 45 & 1 & 4 \\ 0 & 6 & 43 & 0 \\ 0 & 0 & 0 & 54 \end{bmatrix} \begin{bmatrix} 51 & 0 & 0 & 0 \\ 1 & 45 & 3 & 2 \\ 0 & 6 & 43 & 0 \\ 1 & 1 & 1 & 51 \end{bmatrix} \begin{bmatrix} 50 & 1 & 0 & 0 \\ 5 & 38 & 8 & 0 \\ 0 & 6 & 43 & 0 \\ 0 & 6 & 12 & 36 \end{bmatrix} \begin{bmatrix} 51 & 0 & 0 & 0 \\ 3 & 42 & 3 & 3 \\ 0 & 8 & 40 & 1 \\ 0 & 1 & 0 & 53 \end{bmatrix}$$

All of these classified the benign images well, but they are all a little wrong for grades 3 and 4. And for grade 5, the 31x31 Network has a quite poor result, while the others have better results.

Group 4 Group 4 had two Networks that were trained on the full dataset. The error rates for these two is shown in Table 5.4.

Table 5.4: Error rates for the Networks in group 4.

Filter size	Error rate
29x29	15.6%
31x31	20.5%

These error rates are higher than those for group 3, but slighly lower than those for group 2. The confusion matrices for these can be seen below, 29x29 to the left and 31x31 on the right.

$$\begin{bmatrix} 50 & 1 & 0 & 0 \\ 5 & 29 & 14 & 3 \\ 0 & 4 & 45 & 0 \\ 0 & 2 & 3 & 49 \end{bmatrix} \begin{bmatrix} 48 & 3 & 0 & 0 \\ 4 & 31 & 15 & 1 \\ 0 & 6 & 43 & 0 \\ 0 & 1 & 12 & 41 \end{bmatrix}$$

The Networks in this group are quite bad at classifying the grade 3 images. They are also not very good at the grade 4 and 5 images either.

5.2.2 On Validation Data

This Section will show the results when just looking at the classification on the validation images. The Networks are the same as in the previous section.

Group 2 The classification on the validation images for group 2 can be seen in Table 5.5. Below are the confusion matrices for the 19x19 and 25x25 networks.

$$\begin{bmatrix} 6 & 0 & 0 & 0 \\ 1 & 3 & 2 & 0 \\ 0 & 0 & 5 & 1 \\ 0 & 0 & 0 & 6 \end{bmatrix} \begin{bmatrix} 6 & 0 & 0 & 0 \\ 2 & 3 & 1 & 0 \\ 0 & 2 & 4 & 0 \\ 0 & 1 & 0 & 5 \end{bmatrix}$$

Table 5.5: Error rates on validation data for the Networks in group 2.

Filter size	Error rate
17x17	20.8%
19x19	16.7%
23x23	16.7%
25x25	25.0%
27x27	20.8%

Group 3 Table 5.6 shows the error rates for the classification on the validation images for group 3. Below are the confusion matrices for the Networks with filter size 17x17, 29x29, 31x31 and 35x35, in that order.

$$\begin{bmatrix} 6 & 0 & 0 & 0 \\ 1 & 3 & 1 & 1 \\ 0 & 2 & 4 & 0 \\ 0 & 0 & 0 & 6 \end{bmatrix} \begin{bmatrix} 6 & 0 & 0 & 0 \\ 1 & 3 & 2 & 0 \\ 0 & 1 & 5 & 0 \\ 1 & 0 & 0 & 5 \end{bmatrix} \begin{bmatrix} 6 & 0 & 0 & 0 \\ 2 & 2 & 2 & 0 \\ 0 & 1 & 5 & 0 \\ 0 & 2 & 0 & 4 \end{bmatrix} \begin{bmatrix} 6 & 0 & 0 & 0 \\ 2 & 3 & 1 & 0 \\ 0 & 1 & 5 & 0 \\ 0 & 1 & 0 & 5 \end{bmatrix}$$

Table 5.6: Error rates on validation data for the Networks in group 3.

Filter size	Error rate
17x17	20.8%
19x19	25.0%
21x21	29.2%
23x23	25.0%
25x25	25.0%
27x27	25.0%
29x29	20.8%
31x31	29.2%
33x33	20.8%
35x35	20.8%

Group 4 Table 5.7 shows the error rates for the classification on the validation images for group 4. Below are the confusion matrices for both these Networks, 29x29 on the left and 31x31 on the right.

$$\begin{bmatrix} 6 & 0 & 0 & 0 \\ 2 & 0 & 4 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 2 & 0 & 4 \end{bmatrix} \begin{bmatrix} 5 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 1 & 1 & 4 \end{bmatrix}$$

Table 5.7: Error rates on validation data for the Networks in group 4.

Filter size	Error rate
29x29	33.3%
31x31	20.8%

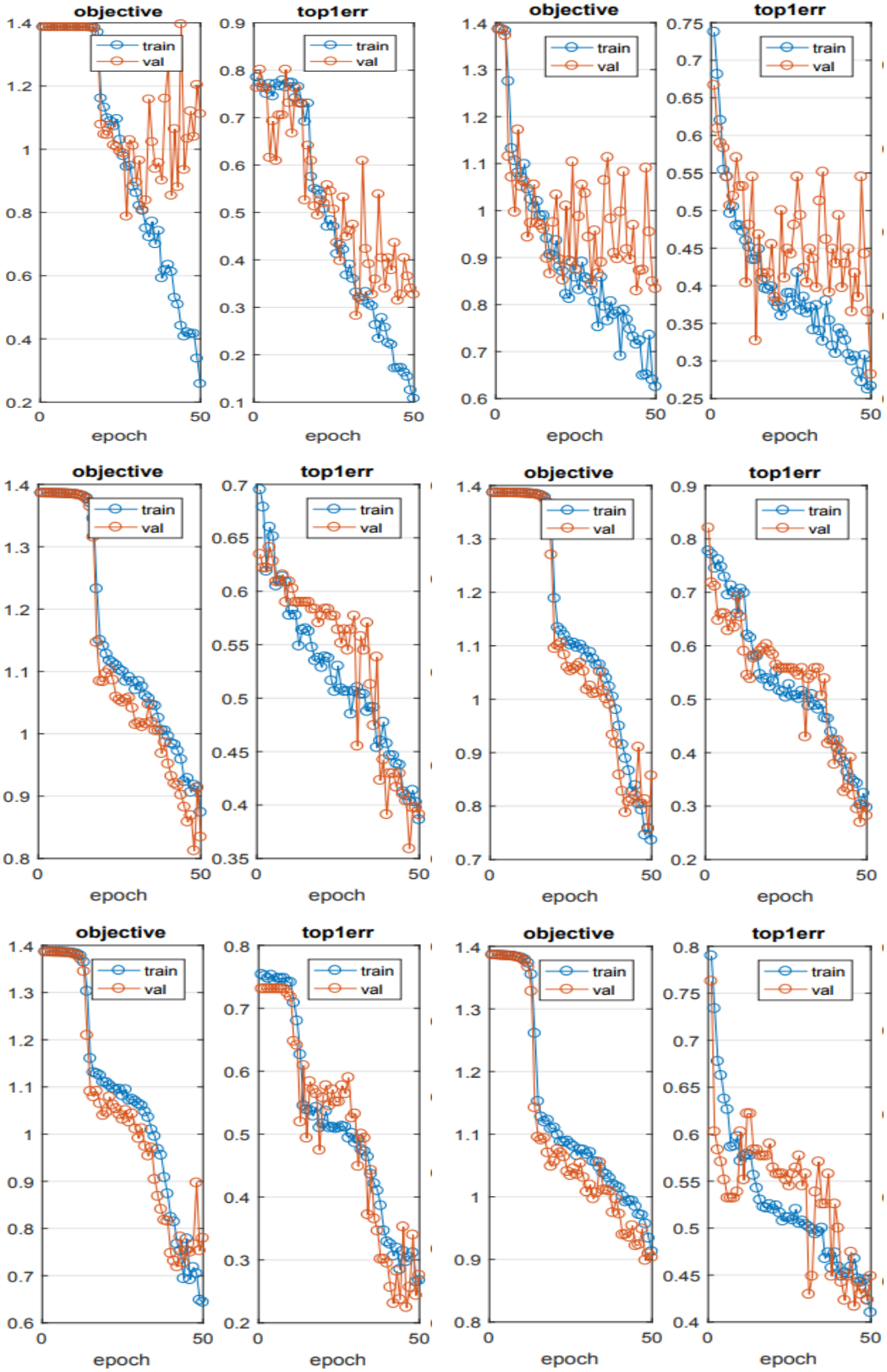

Figure 5.1: Resulting objective function and error over 50 epochs for six of the 19 Networks trained on the smaller dataset. Upper left has filter size 3x3, from group 1. Upper right has filter size 19x19, from group 2. Middle left has filter size 19x19, from group 3. Middle right has filter size 27x27, from group 3. Lower left has filter size 35x35, from group 3. Lower right has filter size 29x29, from group 4.

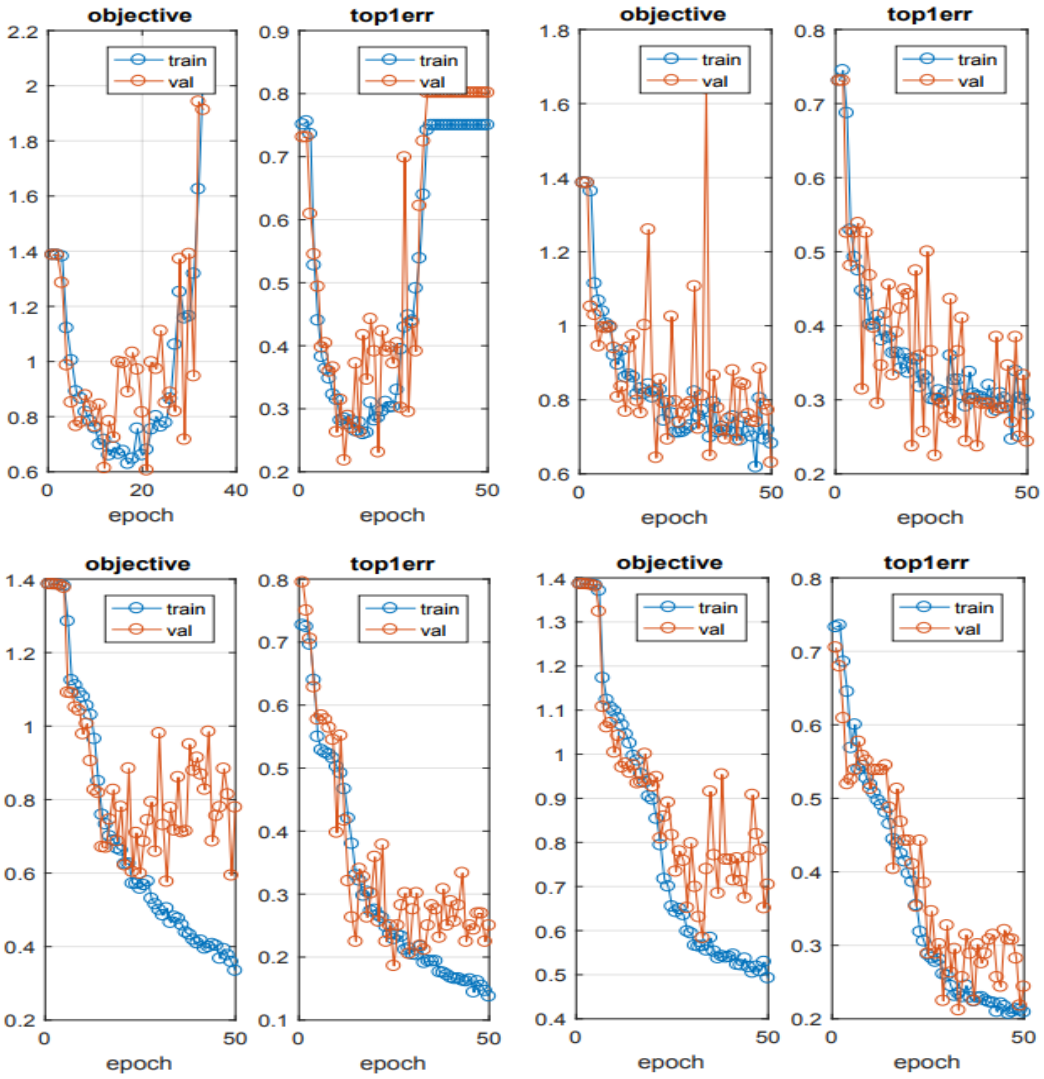

Figure 5.2: Resulting objective and error over 50 epochs for one example from each parameter-group, trained on the full dataset. Upper left: Group 1 with filter size 5x5. Upper right: Group 2 with filter size 17x17. Lower left: Group 3 with filter size 21x21. Lower right: Group 4 with filter size 31x31.

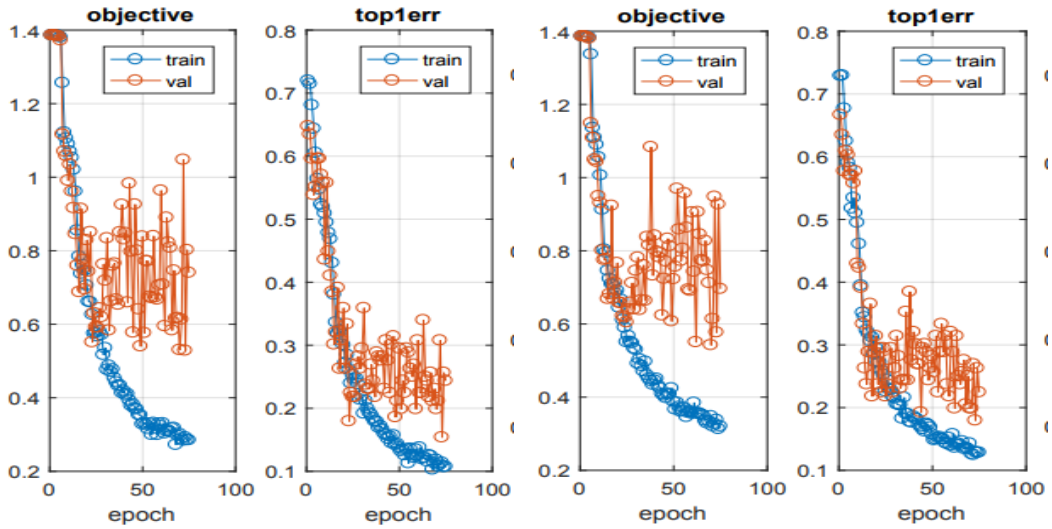

Figure 5.3: Resulting objective and error over 75 epochs where the parameters for the first 50 epochs were as in group 3, and for the last 25 epochs the weight decay was set to 0.1. Left: Filter size 17x17. Right: Filter size 25x25. The Networks were trained on the full dataset.

6

Discussion

6.1 Classification on Patches

The first training that was performed on the smaller subset of the data on all the 102 different Networks showed that the interpolation of the filters most likely plays a role in the result. There were two surprising Networks that showed somewhat promising results from this training, those were the Networks with 3x3 and 5x5 rotating filters from parameter group 1. However, as you can see in the upper left graphs in Figure 5.1 in the objective the validation has started to move upwards which is not a promising sign. I still chose to test these further since the training objective and error went lower than for any of the other Networks, and also because I wanted to see if a larger dataset would reduce the overtraining. The upper left graph in Figure 5.2 shows that this simply did not work.
All of the other Networks that qualified according for further testing had rotating filters of size 17x17 up to 35x35, which was the largest filter size that was tried.
Looking at the resulting objective and error in Figure 5.2, discarding the upper left, there are some differences. The upper right graphs were from a Network from group 2, meaning it had a learning rate of 10^{-3} and a weight decay of 0.1, the lower had a learning rate of 10^{-4} and the lower left had no weight decay, the lower right had 0.1 weight decay. According to the theory, see Section 2.4.4, the upper right and lower right should not be overtrained, or at least not much overtrained since they are trained with a weight decay. The upper right is clearly not overtrained, the validation follows the training well, although the validation error is a bit up and down and the validation objective has some strange peaks. So the weight decay works well for that Network, but the error seems to converge already at 0.3. The lower right also had weight decay, but was still overtrained. The lower right could be compared to the lower left which had the same learning rate. Comparing the objective- and error-values at the last epoch, it does indeed seem like the Network with weight decay is slightly less overtrained. But it also seems like this comes at the price of larger objective- and error-values on the training data. The training error at the last epoch in the lower left graph is around 0.15, while in the lower right graph it is around 0.2. But when looking at the validation errors they are quite similar, with maybe a slightly lower trend in the lower left graph.
The comparison that was just made was between two specific results inside the groups 3 and 4, but the results were similar within the groups so it is still a fair comparison. While the Networks in group 4 were less overtrained, they also needed

larger rotating filters to produce interesting results as seen in Table 5.1. Larger filters means there are more parameters to learn and that it will take longer to train the Networks.
So if one were to look at the results from the 19 Networks in Table 5.1 it seems like the Networks trained with parameters from group 3 were the most interesting. Since they were the most interesting, some of them were trained for a further 25 epochs with an added weight decay of 0.1 in an attempt to reduce the overtraining.. The resulting objective and error for two of these are shown in Figure 5.3. What is wanted there is for the validation objective- and error-values to come down closer to the training objective- and error-values. For the objectives this cannot be seen, but there might be a slight trend towards the training error in the validation errors, especially in the right graph.

6.2 Classification on Full Images

The results from the classification on the full images are in the end more interesting than the result on the extracted patches.
As explained in the method section, the results were obtained by taking the full images and taking out patches spread out over the whole image and then classifying the patches and the final classification of the image will be the most common classification on the patches for the image. Since you want to capture the whole image in patches there will most likely be overlaps between the patches. The only situation in which there would be no overlap would be if all images had both width and height as a multiple of 128 pixels since that is the size of the patches.
The error rates presented in the result is first from classification using both the training images and the validation images combined, and then for just the validation images. Ideally there would have been a third separate test set, but the amount of data was too small for this. The reason for not looking at just the validation images is the same, there were only six validation images for each class. This means that the results from the classification using both the training images and the validation images combined include images that the Network has seen and updated itself based on, which lowers the error rate unless there is absolutely no overtraining.

6.2.1 Results on Training and Validation Data

The results from the group 1 Networks were poor and classified all the images as benign, so those can be discarded.
For the five different Networks in group 2 the error rates ranged from 18% to 33% as seen in Table 5.2. This is not a very good error rate, the best one means that almost one in five diagnoses is wrong which is of course not an acceptable result. Looking at just the best one, the confusion matrix shows that it still classified all benign correctly and only misclassified two of the grade 5 images. But for grade 3 and 4 there were too many misclassifications.
The Networks in group 3 were more successful overall. There were 10 Networks that were promising enough to train on the full dataset and as can be seen in Table 5.3 the error rates are lower for this group than for group 2, except for the 31x31 Network which is a bit of an outlier. And as described in the previous section, these Networks were a bit overtrained so seven of them were trained for another 25 epochs

with added weight decay. These that were trained longer have a lower error rate, but just looking at that does not tell too much since it would be entirely possible for them to have lower error rate if they were trained for longer even without the weight decay. The lowest error rate here was for the 17x17 Network at 6.3%, which is a fairly good result. The confusion matrices show that mostly it is hard to classify the grade 3 and 4 images.
Group 4 had two Networks that were trained on the whole dataset and the error rates for these at 15.6% and 20.5% (Table 5.4) were not as good as group 3. For the Network with 15.6% error rate the confusion matrix shows that the Network has problems especially with classifying grade 3 images.

6.2.2 Results on Validation Data

The results from group 1 does not need to be presented since the validation data was included in the result from both training and validation data which classified all the images as benign.
Group 2 had a lowest error rate of 16.7% and highest at 25.0%. The error rate of 16.7% means that four of the 24 images were misclassified, and 25.0% error means that 6 of the 24 images were misclassified. This straight away shows a problem with representing the results on such a small amount of data as error rates. The error rates differ by quite a bit with just one more or less image misclassified. Comparing with the error rates from Table 5.2 they are around the same values. Looking at the confusion matrices, it seems like it is mostly the grade 3 and grade 4 images that are generally harder to classify.
Group 3 had a lowest error rate of 20.8% and the highest was 29.2%. This is a much higher error rate than when looking at both training and validation data in Table 5.3, where the lowest was 6.3%. This indicates that the Network was overtrained, which was also seen in Figures 5.2 and 5.3. To have an error rate of around 6.3% on a dataset with only 24 images, only one or two images can be misclassified. But for an error rate of 20.8%, five images are misclassified. In the confusion matrices it seems like the grade 3 and grade 4 are still difficult to classify, but there also seems to be a slight problem with grade 5 images - for example the 29x29 Network classified one grade 5 image as benign.
Group 4 only had two Networks and they had error rates of 33.3% and 20.8%. 20.8% is in the same area as the others, while 33.3% means eight misclassified images which is slightly higher than the other presented results. The 29x29 Network that had 33.3% error did not manage to classify any grade 3 image correctly, as can be seen in the confusion matrix, but could classify all the benign and grade 4 images correctly.

6.2.3 Comparison with Regular Convolutional Neural Network

The dataset used in this thesis has been used in another paper which used regular Convolutional Neural Networks to classify the images, see [14]. That paper expanded the dataset with rotations of the images, and trained a regular CNN with more than 12 000 patches for each class. That Network achieved an error rate of 7.3%, which is on only validation data. That error rate is comparable to the best error rates when

looking at both training and validation data in this thesis, but when looking at just the validation data the best error rate was 16.7% or four out of 24 misclassified.
Some things to consider while comparing this thesis with the paper are that [14] focused on creating one Network to achieve a lowest possible error rate and they used cross validation to get the final error rate. Since the focus in this thesis has been more on investigating different filter sizes in the Networks, there has not been time to do this cross-validation, since the training of all the Networks took more than one week. Designing and optimizing one specific Network has also not been done here.

6.3 The Filters

Since a big focus has been on the sizes of the rotating filters, it could be interesting to see what the filters might look like for some Networks. However, as will be seen in the coming Figures, it's hard to interpret what they really do and why they have evolved to what they are.
Figure 6.1 shows the filters for the 19x19 Network with error rate 18.0% from group 2. There the general trend is for a brighter center of the filter, and also many of the filters appear to be green, even though the images are more pink or purple.
Figure 6.2 shows the filters for the 17x17 Network which had the lowest error rate of all Networks at 6.3%. These appear to have barely any structure to interpret at all. This could indicate overtraining of the Network, which has been mentioned earlier. Some have a brighter center while some have a darker center.
Figure 6.3 shows the filters for the 35x35 Network in group 3 that had an error rate of 9.3%. These are of course a bit more detailed since they have around four times as many pixels as the earlier two. Some of these appear to have some kind of cyclic symmetry to them, with alternating brighter and darker going out from the center, but they still look quite noisy.
Figure 6.4 shows the filters for the 29x29 Network in group 4 with error rate 15.6%. Some of these also have the cyclic symmetry appearance that some had in figure 6.3, and they also seem slightly less noisy than those. The colours of the filters also seem more pink which was the colour of the data.
That the filters from group 3 are more noisy than those from group 2 and 4 most likely has something to do with them being trained without weight decay for the most part.
Comparing the filters achieved in this thesis with the filters in [4], those filters have a more directional pattern, while the filters in this thesis had a more circular pattern not unlike the filters achieved in [4] when using a regular CNN with the dataset extended with rotations of the images. However, the dataset used in [4] contained images that had a clearly directional pattern themselves, so the comparison between those filters and the filters from this thesis does not need to mean that the filters in this thesis are somehow wrong.

6.4 Conclusion

This thesis investigated the usage of rotation invariant Convolutional Neural Networks to classify microscopy images of prostatic tissue. Since the rotation invariant

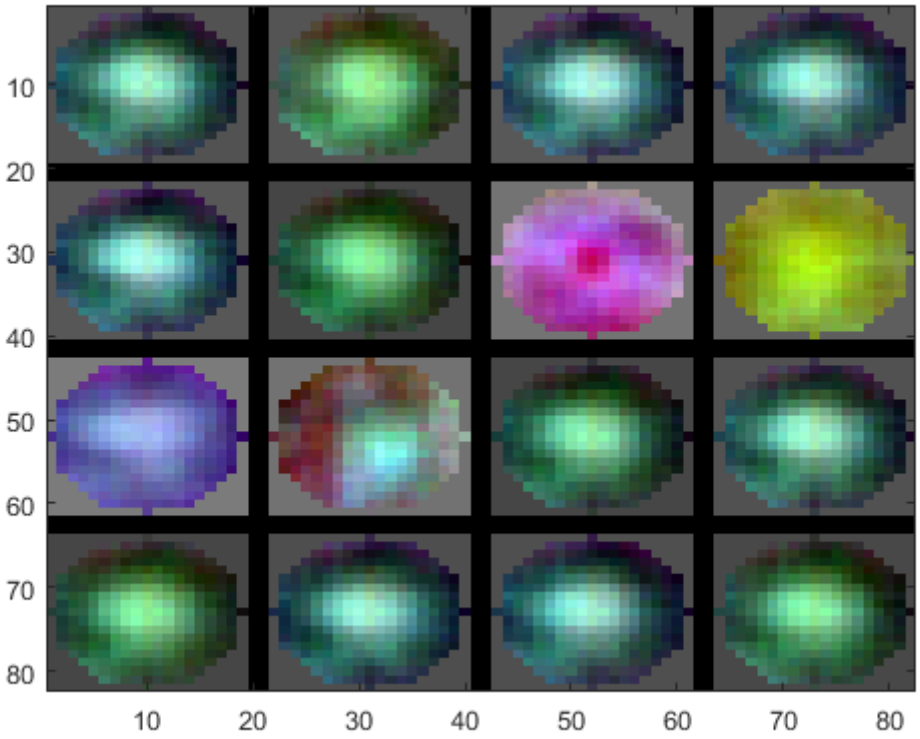

Figure 6.1: The filters in the rotating layer of size 19x19 in group 2.

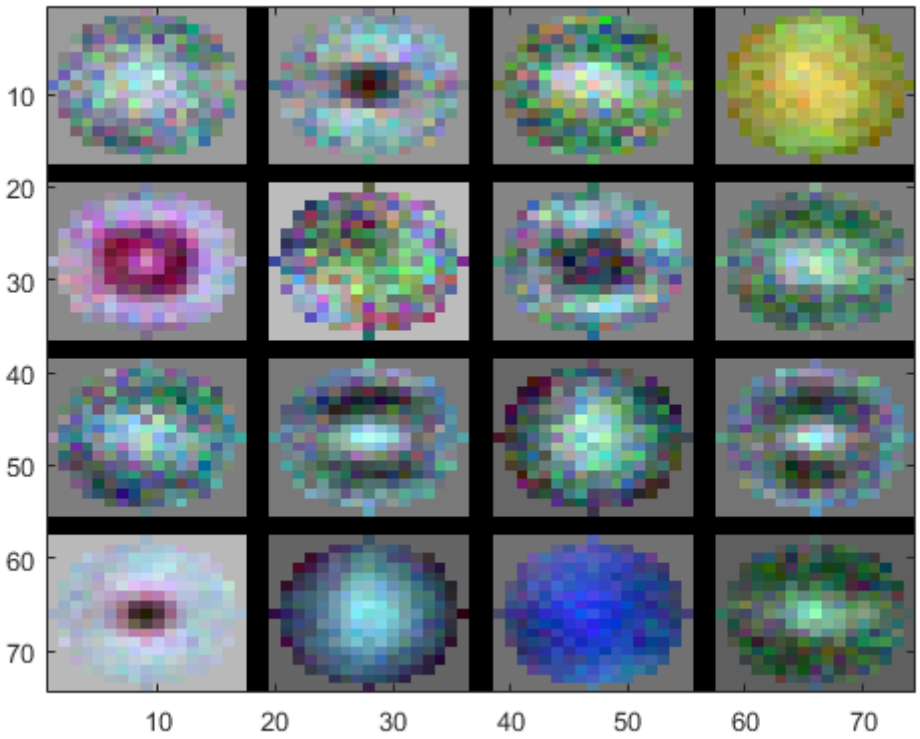

Figure 6.2: The filters in the rotating layer of size 17x17 in group 3.

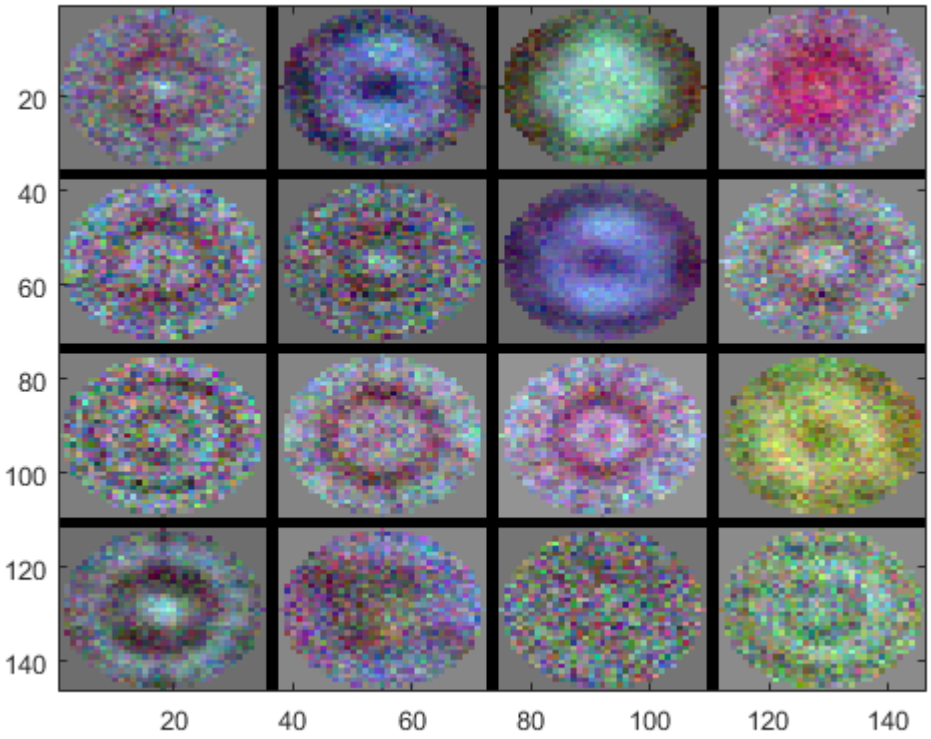

Figure 6.3: The filters in the rotating layer of size 35x35 in group 3.

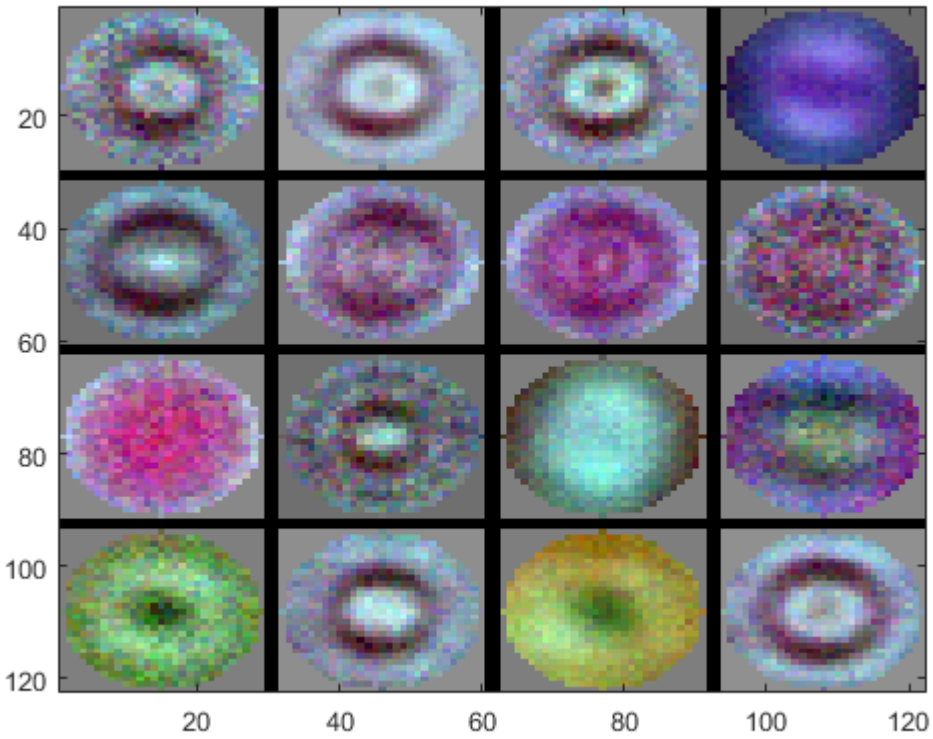

Figure 6.4: The filters in the rotating layer of size 29x29 in group 4.

Networks make use of rotating filters, the problem of interpolation arises. The tests performed showed that interpolation indeed is an issue when rotating the filters. The Networks that showed results that were at least interesting enough to investigate further had a filter size of at least 17x17 in the layer with rotating filters, except for the two Networks with 3x3 and 5x5 filters which in the end proved to be poor sizes. The error rates for the classification on both training and validation data over all Networks ranged from 6.3% up to above 30%. An error rate of 6.3% is approximately the same as saying that 1 in 15 images are misclassified. This is not good enough to use in a real setting. That result is also boosted considering the classification that achieved those low error rates on the full images was done on the whole dataset, including the training set.
The error rates when just looking at the validation images ranged from 16.7% up to 33.3%. This result is worse than for the error rates on both training and validation data, which is to be expected. Since the validation data only consisted of 24 images the error rates change by approximately four percentage points when one more or one fewer image is misclassified. The error rate 16.7% means that four out of the 24 images were misclassified, or 1 in 6. This is simply not a good enough result to apply these Networks in a real setting.
In the comparison with a regular Convolutional Neural Network it seems like the regular Network is better right now, but one should keep in mind that the Networks in this thesis were most likely not the optimal networks since the testing was done in the most general way possible. So it should be possible to reduce the error rate with Networks that are more optimized for the task. It should also be noted that the training for regular Networks is done with a much more expanded dataset, since it can be augmented with rotations of the images.

6.5 Future Work

There are possibilities to expand on the work done in this thesis in many ways. The Networks were not designed specifically for a certain size of rotating filters, but instead used zero-padding to be able to compare the different sizes. So one could try to design Networks specifically for some filter size that were determined to produce good results. It is of course also possible to adjust parameters to enhance the results, one possibility is to use an adaptive learning rate. It would also be preferable to have more data.

Bibliography

[1] Jacques Ferlay, Isabelle Soerjomataram, Rajesh Dikshit, Sultan Eser, Colin Mathers, Marise Rebelo, Donald Maxwell Parkin, David Forman, and Freddie Bray. Cancer incidence and mortality worldwide: Sources, methods and major patterns in globocan 2012. *International Journal of Cancer*, 136(5):E359–E386, 2015. ISSN 1097-0215. doi: 10.1002/ijc.29210. URL http://dx.doi.org/10.1002/ijc.29210.

[2] Socialstyrelsen. Statistik om nyupptäckta cancerfall 2015. 2017. URL http://www.socialstyrelsen.se/Lists/Artikelkatalog/Attachments/20462/2017-1-14.pdf.

[3] Skriande brist på patologer skapar kris i cancervården. http://www.dn.se/debatt/skriande-brist-pa-patologer-skapar-kris-i-cancervarden/, 2015. Accessed: 2017-08-30.

[4] Diego Marcos Gonzalez, Michele Volpi, and Devis Tuia. Learning rotation invariant convolutional filters for texture classification. *CoRR*, abs/1604.06720, 2016. URL http://arxiv.org/abs/1604.06720.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[6] Deep learning for complete beginners: convolutional neural networks with keras. URL https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html. Accessed: 2017-09-27.

[7] Léon Bottou. On-line learning in neural networks. chapter On-line Learning and Stochastic Approximations, pages 9–42. Cambridge University Press, New York, NY, USA, 1998. ISBN 0-521-65263-4. URL http://dl.acm.org/citation.cfm?id=304710.304720.

[8] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus Robert Müller. *Efficient BackProp*, pages 9–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN 978-3-540-49430-0. doi: 10.1007/3-540-49430-8_2. URL https://doi.org/10.1007/3-540-49430-8_2.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[10] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[11] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL http://arxiv.org/abs/1207.0580.

[12] Donald F. Gleason and George T. Mellinger. Prediction of prognosis for prostatic adenocarcinoma by combined histological grading and clinical staging. *The Journal of Urology*, 167(2):953 – 958, 2002. ISSN 0022-5347. doi: http://dx.doi.org/10.1016/S0022-5347(02)80309-3. URL http://www.sciencedirect.com/science/article/pii/S0022534702803093.

[13] 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, James A. Eastham, Peter Wiklund, Misop Han, Chandana A. Reddy, Jay P. Ciezki, Tommy Nyberg, and Eric A. Klein. A contemporary prostate cancer grading system: A validated alternative to the gleason score. *European Urology*, 69(3):428 – 435, 2016. ISSN 0302-2838. doi: http://dx.doi.org/10.1016/j.eururo.2015.06.046. URL http://www.sciencedirect.com/science/article/pii/S0302283815005576.

[14] Mattias Ohlsson Niels Christian Overgaard Agnieszka Krzyzanowska Anders Heyden Anders Bjartell Kalle Aström Anna Gummeson, Ida Arvidsson. Automatic gleason grading of h and e stained microscopic prostate images using deep convolutional neural networks. *Proc.SPIE*, 10140:10140 – 10140 – 7, 2017. doi: 10.1117/12.2253620. URL http://dx.doi.org/10.1117/12.2253620.

Master's Theses in Mathematical Sciences 2017:E64
ISSN 1404-6342

LUTFMA-3334-2017

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

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