DEGREE PROJECT IN MATHEMATICS,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2018

Classification of Lung Tumors by using Deep Learning

JOHANNA BERGER

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ENGINEERING SCIENCES

Classification of Lung Tumors by using Deep Learning

JOHANNA BERGER

Degree Projects in Optimization and Systems Theory (30 ECTS credits)
Degree Programme in Applied and Computational Mathematics (120 credits)
KTH Royal Institute of Technology year 2018
Supervisors at Siemens: Oskar Strand
Supervisor at KTH: Johan Karlsson
Examiner at KTH: Johan Karlsson

TRITA-SCI-GRU 2018:248
MAT-E 2018:48

Royal Institute of Technology
School of Engineering Sciences
KTH SCI
SE-100 44 Stockholm, Sweden
URL: www.kth.se/sci

Abstract

The aim of this thesis is to apply deep learning on medical images in order to build an image recognition algorithm. The medical images used for this purpose are CT scans of lung tissue, which is a three dimensional image of a patients lungs. The purpose is to design an image recognition algorithm that is able to differentiate between tumors and normal tissue in lungs. The algorithm is based on artificial neural networks and therefore the ability of a convolutional neural network (CNN) to predict a tumor is studied. Two different architectures are designed in this thesis, which are a three and six layer CNN. In addition, different hyper-parameters and optimizers are compared in order to find suitable settings.

This thesis concludes that no significant difference exists between the results of the two architectures. The architecture with three layers is faster to train and therefore 100 trainings with same settings are completed with this architecture in order to get statistics of the trainings. The mean accuracy for the test set is 91.1% and the standard-deviation for the test set is 2.39%. The mean sensitivity is 89.7% and the mean specificity is 92.4%.

Klassificering av lungtumörer genom tillämpning av djupinlärning

Sammanfattning

Syftet med denna rapport är att utvärdera tillämpningen av djupinlärning på medicinska bilder för att konstruera en bildigenkänningsalgoritm. För detta ändamål används CT-bilder av lungvävnad, vilket är en avbildning av en patients lungor i tre dimensioner. Avsikten är att konstruera en bildigenkänningsalgoritm som är kapabel att differentiera mellan tumörer och normal vävnad i lungor. Algoritmen baseras på artificiella neuronnätverk och därför studeras ett faltande neuronnätverks förmåga att prediktera en tumör i lungvävnad. Vidare utvärderar denna rapport två olika arkitekturer - faltande neuronnätverk av tre respektive sex lager. Slutligen optimeras algoritmen med avseende på hyper-parametrar och optimeringsmetoder.

Denna studie konkluderar att det inte råder någon signifikant skillnad mellan resultaten för de två arkitekturerna. Arkitekturen med tre lager går snabbare att träna och därför är 100 träningar med samma inställningar genomförda med denna arkitektur för att erhålla statistik över träningarna. Medelvärdet för noggrannheten för testdata är 91.1% med en standardavvikelse om 2.39 %. Medelvärdet för känsligheten är 89.7% och medelvärdet för specificiteten är 92.4%.

Acknowledgements

I would like to thank Lena Blom and Oskar Strand at Siemens Healthineers for giving me the opportunity to carry out a interesting and useful project. I especially would like to acknowledge my supervisor Oskar Strand for his time and commitment in guiding and assisting me with this thesis. Finally, I would like to thank Johan Karlsson, my supervisor at KTH, for the support and guidance during this thesis.

Contents

1 Introduction

With progress in health care and medical technology and with the population aging and growing, the demand for health care is increasing. Leading to queues for patients to be diagnosed and increasing strain on the well fare system. Longer queues may lead to patients being diagnosed in later stages of the disease, causing suffering and increased risk of mortality, which in turn requires more resources and spending in the health care sector. But are more resources and more money budgeted to the health care sector the solution? What if the health care sector were digitalized in a wider extent, and what if there would be autonomous diagnosis systems to assist the radiologists? Today, artificial intelligence (AI) and self-learning systems are widely discussed in the health care sector, rendering both curiosity and skepticism about these future technologies. But it is a fact that the health care sector is in need of innovative and modern solutions to become more efficient.

Cancer is a common disease, and lung cancer is the most common cancer diagnose in the world. It is crucial for the survival of a patient to be diagnosed in an early stage of the cancer. The earlier the cancer is detected and treated, the higher the survival-rate of the patient. The diagnosis of lung cancer is often based on images of the lungs, such as conventional X-ray system and CT scans. It is a difficult and time consuming task to interpret medical images, and to diagnose a patient using images an experienced radiologist is needed. By implementing autonomous diagnosis systems as decision support for radiologists, both costs for the health care sector and patients suffering can be reduced. The aim of this thesis is therefore to investigate the possibility of a classification algorithm to differentiate between lung tumor tissue and normal lung tissue in CT scans.

1.1 Background

Lung cancer is the most common cancer diagnose in the world and the fourth most common cancer diagnose in Sweden. In both the world and Sweden lung cancer is the type of cancer that causes most deaths. The major cause of lung cancer is smoking and up to 90 % of those who are diagnosed with lung cancer are current or previous smokers [3].

There are several stages of cancer, and a patient is diagnosed in a certain stage depending on if or how extensively the cancer has spread or metastasized. It is more difficult to detect an early stage of lung cancer, since there are fewer symptoms for the early stage. A patient has early stage cancer if there is a tumor, an abnormal growth in the lung tissue, less than 10 mm in diameter in the lungs [6]. A tumor can be benign or malignant, i.e. non-cancerous or cancerous respectively. A malignant tumor means it is able to metastasize, leading to a lower survival-rate for a patient. The larger the lung tumor is the more likely it is to be malignant.

1.1.1 Diagnosis of lung tumors using imaging systems

When a patient is suspected of having lung cancer, the first arrangement is to X-ray the lungs to obtain a two dimensional image. In most cases cancer is detected using X-ray, where both

location and size can be determined. But there are, however, difficulties detecting tumors with a diameter less than one centimeter using X-rays.

It is common to arrange an additional examination in adjunct to the X-ray, i.e. a computed tomography scan (CT scan). During the examination, the patient is lying on a bunk in a circular opening of the machine and an image can be computed using X-rays. A CT scan produces cross-sectional images, also referred to as slices, which generates a three dimensional image of the patient. A CT scan can detect smaller tumors than X-ray imaging and this examination is often performed in order to determine if the cancer has metastasized [4].

Figure 1 shows two different slices of two different CT scans, notice that the sizes of the lungs vary in the images. The reason for this is that the slices are in different axial positions of the lungs.

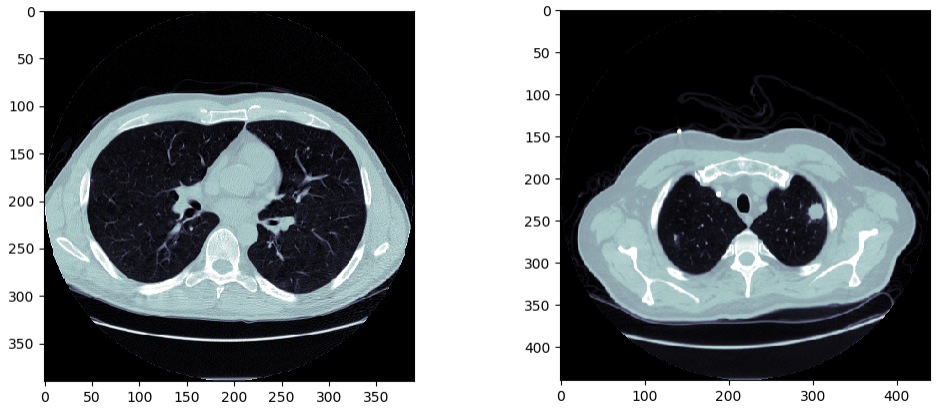

Figure 1: Two slices of CT scans from two different patients. The image to the left is not containing a tumor and the one to the right contains a large tumor with a diameter of 21 mm.

Detecting lung tumors, especially smaller ones, in a CT scan is a difficult assignment for the radiologists since normal structures are hardly distinguishable from tumors. This is because of lymph nodes, nerves, muscles and blood vessels look very much alike the tumor growth in CT images [6]. Even the most experienced eye has trouble finding the small tumors in a CT scan, which in addition normally consist of 400 slices of 512×512 pixel images of the whole lungs. Finding tumors is thus time consuming, also due to the fact that the structure of the lungs makes it difficult to distinguish small tumors.

1.1.2 Autonomous diagnosis of lung tumors

Autonomous diagnosis can reduce patients queuing and make the diagnosis process more efficient. The diagnostic process could be shortened by implementing autonomous diagnosis systems and time can be saved for radiologists leading to reducing the suffering for patients. The intention of implementing the autonomous diagnosis system is to provide the radiologists with a decision support, with a purpose of speeding up the diagnostic process and

preventing diagnostic errors. This tool can assist the radiologists in meeting the rising demand of diagnostic imaging.

The power of AI has shown promising results, but it has still not completely reached the health care sector. Imaging is a common diagnostic method and by applying deep learning on medical images a classifying system can be achieved. In other words, an image recognition algorithm that differentiates between tumors and non-tumors can be designed. The ability of classifying tumors in CT scans of lungs can thus be implemented and used as an autonomous diagnosis system of simpler cases. The more complicated cases may require more examinations, but the optimal solution would be if the algorithm could be able to classify cases that are complicated for radiologists to diagnose.

1.1.3 Image recognition algorithm

In this thesis the image recognition algorithm being designed is based on deep learning, more precisely artificial neural networks. Both neural networks and convolutional neural networks (CNN), a type of artificial neural network, are utilized to build the image recognition algorithms. In previous work, CNN has shown excellent performance for classifying images [5], and is therefore considered as a feasible method to apply on medical images. By letting the network experience the images and providing the true class of the images, it can learn to classify a tumor in lung tissue. Since there are two classes, tumor or non-tumor, it is a binary classification algorithm.

The features of small tumors can be difficult to distinguish from the normal structure of lungs, especially when observing the slices of a CT scan. The slices are images of two dimensions, where a vessel can, e.g., be mistaken as a tumor. By using images of three dimensions, as a complete or parts of a CT scan, the features of the tumors are more distinguishable from the normal structure of lungs. This property is taken into account by designing a CNN that has three dimensional images as inputs.

There are disadvantages of a three dimensional CNN, e.g. the need of computer memory. In comparison with a two dimensional CNN it is computational inefficiency. On the contrary, a two dimensional CNN generates more false positive classifications, i.e. non-tumor classified as tumors, due to the lung structure [5]. For this reason, a three dimensional CNN is preferable.

1.1.4 Siemens Healthineers and artificial intelligence in health care

Siemens Healthineers has been involved in the field of machine learning since the 1990s. This has resulted in more than 400 patents in the field, 80 patent applications in deep learning and more than 30 AI-powered applications. An AI-powered product that Siemens Healthineers has developed is, e.g, an advanced visualization solution called *syngo*.via. In *syngo*.via there is for instance an application called ALPHA (Automatic Landmarking and Parsing of Human Anatomy), which is an automatic contouring tool for a fast segmentation. APLHA is based on a pattern recognition algorithm that automatically detects anatomical structures

in images of different imaging systems. For example, simultaneous 3D visualization of heart valves anatomy and blood flow or the "CT Bone Reading" for virtual unfolding of the rib cage. This is a tool for the radiologists that simplifies the diagnosis through advanced visualizations.

Another AI-based product is the Fully Assisting Scanner Technologies (FAST) 3D Camera, used along with a CT scan for a precise patient positioning. The FAST 3D Camera captures the shape, height and position of the patient using infrared measurement and leaves information about the patients direction, i.e. "head-first versus feet-first" as well as "prone or supine", the body regions in z-direction, table height and patient thickness. This AI-powered solution assists the radiologist to position the patient in a CT scan.

1.2 Aim of thesis

The aim of the thesis is to design an image recognition algorithm that differentiates between tumors and non-tumors, by using CT scans of lungs. The purpose is also to design an algorithm that is able to make decisions and may improve health care, with respect to time efficiency and reducing patient queues.

The classifying algorithm is built by applying deep learning, which means implementing a CNN. A classifying system can be achieved by providing the network images with corresponding classes, letting the network learn through experience and an ability of predicting tumors can thus develop. The aim of the thesis is reached if a CNN is able to classify images to contain a tumor or not.

1.3 Research questions

- Is a three dimensional CNN suitable for a binary classification of lung tumors?
- Can it differentiate between tumor tissue and normal tissue?
- How does the performance depend on hyper-parameters and the architecture?

1.4 Thesis outline

In *Section 1* the subject of the thesis is introduced and the background is discussed, in order to emphasize the importance of the thesis. This section contains arguments for how an image recognition algorithm based on deep learning can make the health care sector more efficient. Further the method used to design the image recognition algorithm is introduced and motivated. The method used is based on convolutional neural networks. Finally, the aim of thesis and the research question are presented.

Section 2 contains the theory needed in order to design an image recognition algorithm based on convolutional neural network. The theory includes the different operations in a convolutional neural network and the optimization process.

In *Section 3* the method used to design an image recognition algorithm is introduced. The method is divided in two parts, i.e. image preprocessing and classification, and the details of the implementations are presented in this section.

Section 4 includes the results of the image recognition algorithm, which are presented in tables and several plots.

In *Section 5*, the results are discussed and conclusions are drawn. The discussion also contains thoughts about future work, improvements and applications.

2 Theory

In this chapter the theory needed to build a binary image classification algorithm is discussed and the aim is to give the reader relevant information about the building blocks of such an algorithm. Deep learning is commonly used in image recognition, since it has shown success. In deep learning an algorithm learns from experience of input data, where the algorithms are defined by networks built of a large number of parameters. The adjustable parameters are optimized by providing the true class of the input image, i.e. the label, in order to recognize its typical patterns. Supervised learning is when the networks are optimized with respect to a provided label and the optimization process is usually called training.

There are several deep learning architectures, and in this thesis neural networks and CNNs are treated. Neural networks and CNNs in combination are commonly used in order to design an image recognition algorithm, where the CNN initiates with convolutional operations and finishes with the structure of neural networks. The CNN processes the image with the intention of finding typical features, and the neural network is used to reduce the parameters into the number of classes that the images can belong to. The elements of the output vector are the probabilities that correspond to the likelihood of the input belonging to the different classes and with this information a classification of the input can be accomplished.

2.1 Artificial neural network

An artificial neural network (ANN) is inspired by the functionality of the human brain, where billions of interconnected neurons process information [17]. A brain learns to recognize objects by using the human senses and impressions, while the ANN is learning by using functions stacked on top of each other. In other words, the brain is imitated by functions that are mapping an input set to the corresponding output set.

2.1.1 The perceptron

The simplest case of an ANN is the perceptron, which consist of one neuron and is illustrated in Figure 2.

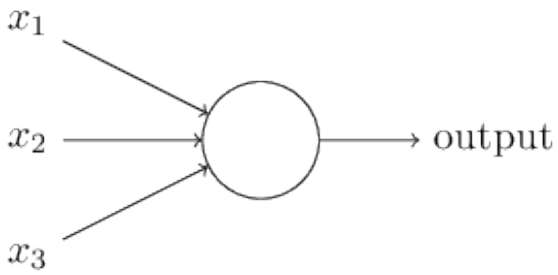

Figure 2: An artificial neuron with inputs x_1, x_2, x_3[14], without the weights and bias for simplicity.

The perceptron has weighted inputs, a bias and an output that is defined by a non-linear function. The input of the non-linear function is the summed weighted input data and an added bias [14], thus the output can be defined by

$$y = f(\sum_{k=1}^{K} w_k x_k + b). \tag{1}$$

The number of inputs corresponds to K, the x_k are the inputs, b is the bias, y is the output and f is the non-linear function. Additionally, the bias corresponds to the difference between the model itself compared to the best hypothetically possible model. For simplicity, b can be set to w_0 and x to 1, then Equation (1) can be rewritten to

$$y = f(\sum_{k=0}^{K} w_k x_k) = f(Wx) \tag{2}$$

$$y \in \mathbb{R} \tag{3}$$

$$W \in \mathbb{R}^{K \times 1}. \tag{4}$$

2.1.2 Non-linearity functions

Section 2.1.1 describes how the output of a perceptron is defined by a non-linear function. The non-linear function is usually called the activation function, since its purpose is to decide whether the outside connections should consider the neuron as activated or not. In neural networks, the sum of the linear functions can assume any real number and if it, e.g., exceeds a certain threshold then the activation function will imply that the neuron is activated and the information will be sent forward. There are several different activation functions and the most common ones are the *tanh* function [7]

$$f(x) = \tanh(x), \tag{5}$$

the sigmoid function

$$f(x) = \frac{1}{1 + e^{-x}} \tag{6}$$

and finally the rectified linear unit (ReLu) function

$$f(x) = \max(0, x). \tag{7}$$

2.1.3 Deep neural networks

In a deep neural network (DNN), several neurons are stacked in layers and are connected such that the outputs from the previous layer are inputs to the next layer. The first layer in a neural network is called the input layer, the final layer the output layer, and the intermediate layers the hidden layers [14]. A neural network is called deep when it consists of one or more hidden layer and a fully connected neural network is when each neuron in the layers are connected.

Figure 3 is a simple example of a fully connected neural network, but is also a DNN that consist of two input nodes, a hidden layer with three neurons and two output classes.

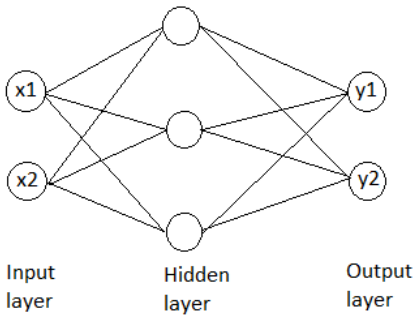

Figure 3: An example of a fully connected neural network, without weights and bias for simplicity.

The example in Figure 3 can be described by

$$y_m = f_{out}(\sum_{l=0}^{3} w_{lm} f_{hid}(\sum_{k=0}^{2} w_{kl}^{'} x_k)). \tag{8}$$

In Equation (8), y_m is the output and $m = 1, 2$ is the number of output classes, where the largest value of y_m is the most probable class for the input x_k. Furthermore, $k = 1, 2$ is the number of inputs and $l = 1, 2, 3$ is the number of hidden neurons. The non-linear activation functions are denoted by f_{out} and f_{hid}. Finally, the w_{lm} and $w_{kl}^{'}$ are the adjustable weights to be trained, where each weight corresponds to a connection between two neurons.

Usually, a DNN consists of more layers than illustrated in Figure 3 and in order to describe the definition of the layers more generally, the layer k is described by

$$y_k = f(W_k y_{k-1}) \tag{9}$$
$$y_k \in \mathbb{R}^{n_k} \tag{10}$$
$$W_k \in \mathbb{R}^{n_k \times n_{k-1}}. \tag{11}$$

The activation function f is element-wise non-linear and produces the output y_k. The trainable weight matrix W_k has dimension $n_k \times n_{k-1}$, where n_k is the number of units in layer k and n_{k-1} is the number of units in the previous layer. If the neural network consist of l layers then $k = 1, ..., l$.

A DNN can be called a feedforward neural network if the initial input x propagates forward through all the layers and finally produces the output $\hat{y}$. Algorithm 1 describes the iterative method to propagate the input data through the network and the method is called forward propagation [11].

The concept of forward propagation is that the output of the previous layer, $h^{(k-1)}$, is multiplied with the weight matrix of the current layer, $W^{(k)}$. This multiplication is equal to the vector $a^{(k)}$, and the elements of $a^{(k)}$ corresponds to the nodes in layer k. Thereafter, the non-linear activation function f is applied, which is element-wise non-linear, and the

outcome, $h^{(k)}$, will be the input for the next layer. This procedure is done iteratively until the last hidden layer $l-1$. After the output layer l another activation function is implemented and is denoted as s. Then the final output $\hat{y}$ is produced. Afterwards, $\hat{y}$ is the input for the objective function, $J(\theta) = l(y, \hat{y})$ described in Section 2.3.1. The parameters, i.e. the weights, are denoted as θ and y is the label for the input sample.

Algorithm 1 Forward propagation [11]

Require: Network depth, l
Require: $W^{(k)}$, $k \in 1, ..., l$, the weight matrices of the model
Require: x, the input to process
Require: y, the label output
$h^{(0)} = x$
for each $k = 1, ..., l-1$ **do**
    $a^{(k)} = W^{(k)} h^{(k-1)}$
    $h^{(k)} = f(a^{(k)})$
end for
$a^{(l)} = W^{(l)} h^{(l-1)}$
$\hat{y} = s(a^{(l)})$
$J(\theta) = l(y, \hat{y})$

The output layer of the neural network returns a vector that consist of as many elements as classes, and it is the input for the activation function s. The activation function s is called softmax and is commonly used after the output layer, as seen in Algorithm 1. The softmax function is defined as

$$\hat{y}_i = \frac{e^{y_i}}{\sum_{j=1}^{n} e^{y_j}}. \tag{12}$$

The softmax function converts each element to a value between zero and one by dividing them with the sum of the elements. Thus, the vector is normalized into positive values that sum to one and will correspond to probabilities of the inputs belonging to each class [1].

However, there are limitations in fully connected neural networks due to the fact that each weight is optimized separately, since it assumes that the weights are independent. For example, if a fully connected neural network detects a typical feature at a certain location of an input image, then the weights corresponding to that activation will be specific to the location of the feature. This means that the detections are spatially dependent and that a feature need to be seen at a particular location in order to be recognized [11]. For this reason the convolutional neural network is discussed in Section 2.2.

2.2 Convolutional neural networks

A convolutional neural networks (CNN) is a network that take advantage of the spatial structure of the input images [14] and a property that is utilized, is that nearby pixel values are more correlated than distant pixel values. The approach is to exploit the dependence of

the nearby pixel values to extract local features, which only depend on small sub regions of the image [8].

A CNN can consist of several blocks, and each block can be described with three stages. The first stage is a layer that produces a set of linear activations by performing several linear operations in parallel. Each linear operation can be seen as a spotlight that illuminates small sub regions of the input image, with the intention of detecting typical features, and the linear operation is called convolution. The second stage is when the activation function is applied, and the input is the linear activation from the convolution. This stage is also called the detector stage, since the output will represent the features that were detected in the input image. Finally the third stage, max pooling, which is an operation that reduces the number of parameters in order to increase the efficiency [7]. The max pooling is reducing the height and the width of the propagating input image, but still keeping the crucial features.

2.2.1 Mathematical formulation of convolution

The networks are called CNN when most of the linear operations are convolutions, instead of ordinary matrix multiplications. In the one-dimensional case, the convolutional operation is described by

$$(x * w)(t) = \int x(a)w(t-a)da. \tag{13}$$

Additionally, the discrete convolution can be written as

$$(x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a). \tag{14}$$

In Equation (13) and (14) both x and w are real valued functions and are often referred as the input function and the kernel. Usually, convolutions over more than one dimension at a time is used and the two-dimensional case is described by

$$(I * K)(i,j) = \sum_{m}^{M}\sum_{n}^{N} I(m,n)K(i-m, j-n). \tag{15}$$

Both I and K are two-dimensional, where I is a matrix and K is the kernel. The kernel K has the dimension $M \times N$ and i, j is the location in the output matrix. Since convolution is commutative it is equivalent to write

$$(K * I)(i,j) = \sum_{m}^{M}\sum_{n}^{N} I(i-m, j-n)K(m,n). \tag{16}$$

The commutative property holds since there are minus signs in the indices and the kernel is flipped relative to the input. Flipping the kernel means flipping the rows upside down and the elements in the rows are flipped left to right [11].

2.2.2 The convolutional layer

In a CNN the adjustable weights are called the kernel and usually for each convolutional layer there are several kernels. The kernel convolves over the input image and transforms it to a feature map. One way to think of it is that each kernel is trained to detect a certain feature, and the feature map is the linear activations of the input image. This indicates that each kernel will detect the same pattern but at different locations in the input image [8] and a complete convolutional layer consist of several kernels that detect different features. For instance, if the kernel is trained so that it can pick out a vertical edge, then it is able to find that feature everywhere in the input image [14].

For a convolutional layer the hidden neurons are not equivalent to the hidden neurons in a DNN, discussed in Section 2.1.3. In a CNN, the elements in the feature map correspond to the hidden neurons. The kernel is therefore called the shared weights, since the hidden neurons have the same weights. For the two-dimensional case, the hidden neuron at location (i, j) in a feature map is defined by

$$y_{i,j} = f\left(\sum_{m=1}^{M}\sum_{n=1}^{N} w_{m,n} y_{i-m,j-n} + b\right) \tag{17}$$

$$y_{i,j} \in \mathbb{R} \tag{18}$$

$$w_{m,n} \in \mathbb{R}^{M\times N}. \tag{19}$$

The scalar $y_{i,j}$ is the non-linear activation for location (i, j) in the feature map, where $i = 1, ..., I$ and $j = 1, ..., J$. Thus, for each (i, j) a hidden neuron is produced and the dimension of the feature map is $I \times J$. Additionally, $y_{i-m,j-n}$ is the activation from the previous feature map or input image, b is the bias and $w_{m,n}$ is the adjustable kernel with dimension $M \times N$.

Figure 4 is an example of how a flipped kernel convolves over an image, resulting in a feature map built of hidden neurons. As discussed in Section 2.2.1, the rows of the kernel, and its elements, are flipped. For example, the marked 8 in the feature map is calculated by using

$$\text{Marked part of the example image} = \begin{pmatrix} 9 & 6 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad \text{Flipped kernel} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{20}$$

Then the 8 is obtained by the sum of the elementwise multiplications

$$1 \cdot 9 + 0 \cdot 6 + 0 \cdot 0 + -1 \cdot 1 = 8. \tag{21}$$

The marked area in the input image is called the receptive field and is the area in the image where the kernel is operating. The kernel is moving over the image with a suitable step length that is called the stride length. The stride length affect the size of the feature map, i.e. the larger stride length the fewer hidden neurons in the feature map [14].

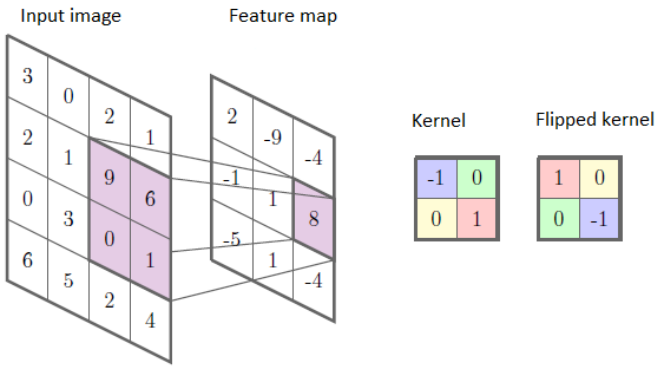

Figure 4: An illustration of how a flipped kernel convolves over an example image, resulting in a feature map [12].

In comparison with the fully connected neural networks, discussed in Section 2.1.3, a CNN has a significantly smaller number of parameters. This is due to the fact that in a CNN a sub region of the input image is connected to one hidden neuron, whereas in a fully connected neural network each pixel value would be connected to each neuron in the next layer [14].

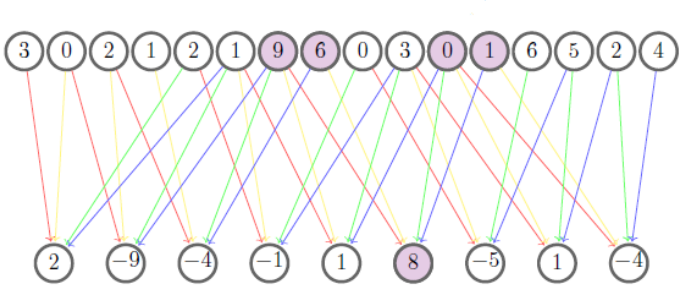

Figure 5: The input nodes corresponds to the entries in the input image in Figure 4. This is an illustration of the connections between the input values and the hidden neurons presented as a sparsely connected ANN [12].

In Figure 5 the top layer represents the input layer, corresponding to the elements in the input image in Figure 4. Further in Figure 5 the marked nodes in the input layer correspond to the elements in the receptive field. Consider the marked number 9 in the input layer and notice that it is only connected to four of nine possible hidden neurons, meaning that the network is sparsely connected. A CNN is therefore said to be a sparsely connected network. To interpret the sparse connectivity consider the connections between the nodes of the input layer and the neurons in the hidden layer, they are less than in a fully connected network.

The sparse connections appear when the kernel is convolving over small sub regions of the input image, and if a pixel value is included in several sub regions then it will have connections to several hidden neurons. The pixel values in the corners, however, are only connected to

one hidden neuron, while a pixel value in the center of the image is connected to four hidden neurons. Due to the sparse connectivity, a CNN is more efficient than a fully connected neural network in image recognition, in regards to computational capacity and memory of the computer analyzing the data. Another essential advantage is that it is difficult to over-fit the CNN since the there are fewer weights to train.

It is common to have several feature maps in one layer. If the input image is two-dimensional it is said to have one channel, then each convolutional operation is figured as a feature map, as illustrated in Figure 6. But if the three feature maps will be an input for a next layer, where the number of feature maps is set to six, then the number of kernels required is $3 \cdot 6 = 18$. Hence, to produce one feature map from an input consisting of three feature maps three 2D kernels are required. For example, the first kernel generates a value from the first feature map, the second kernel generates a value from the second feature map and the same for the third, by summing these three generated values the hidden neuron can be obtained.

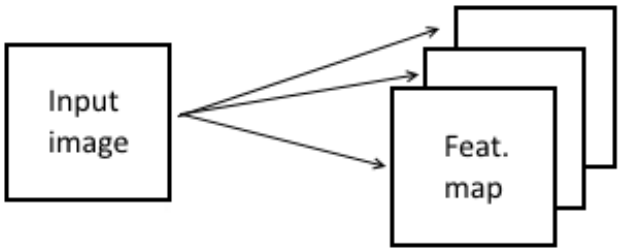

Figure 6: Illustrating how the input transforms to three feature maps in the first convolutional layer. In this example there are three kernels, i.e. three convolutions in parallel.

After each feature map the activation function called Rectified Linear Units function (ReLu) is applied and is described by Equation (7). This activation function will return a feature map consisting of values larger than zero. ReLu has shown to be an efficient activation function during training, with respect to computation time for a CNN [14].

2.2.3 Max pooling

Usually, after the CNN and activation function a max pooling layer is added, that simplifies the feature map by reducing the number of parameters. In max pooling a kernel is used to extract the largest value within the receptive field of the feature map, see Figure 7. If the stride length, i.e. the length that the kernel is moving, is set to two and the kernel has the dimension 2×2, then the feature map will be reduced by a factor of two in both width and height [14].

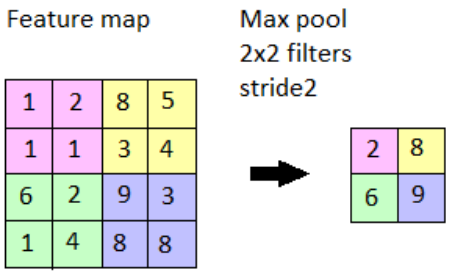

Figure 7: Max pooling with kernel size 2×2 and stride length 2.

A majority of the pooled output values do not change if the input is translated, which means that the representation is approximately invariant to small translations. This property is very useful if it is more important to find the feature than to know exactly where it is [7]. If a feature is detected anywhere in an image then the max-pooling layer will throw away the exact positional information, i.e. once a feature is detected the exact location is not as important as its rough location relative to other features.

2.3 Optimization of the network

The aim of optimizing or training the network is to update the parameters such that the parameters and functions together transform the input data to a number, predicting which class the input most likely belongs to.

In order to train a network, an objective function is required and the value of the objective function is corresponding to a measure of the error between the prediction and label. The parameters are adjusted, with respect to minimizing the objective function, by utilizing gradient-based methods, which in machine learning are called optimizers.

During training, the training set is fed to the network several times, in order to optimize the parameters correctly. When the complete training set has been passed forward and backward through the network, an epoch is performed. The number of epochs has a great influence on the optimization result and too many epochs can cause over-fitting. This is when the network has adjusted the parameters to the training set so much that its ability to generalize to a test set is deteriorated. Fortunately, there are several regularization methods to implement in order to avoid over-fitting during training, such as dropout and early stopping described in Section 2.3.6.

In addition, the training set is often divided in smaller batches since the complete training set is normally too large to feed the network at once. The number of samples in each batch is called the batch size and the division saves memory of the computer.

2.3.1 Objective function

An objective function is required in order to optimize the parameters. The objective function used in this thesis is called the cross-entropy, also referred to as the loss function. The purpose of optimization is to minimize the cross-entropy, i.e. minimizing the difference between the discrete probability distribution of the labels, P, and the distribution of the prediction, Q [2]. This thesis concerns a binary image classification, therefore the binary cross-entropy function is introduced and defined by

$$L(x, y, \theta) = \frac{1}{2} \sum_{i=1}^{2} -y_i \log(\hat{y}_i(x, \theta)) - (1 - y_i) \log(1 - \hat{y}_i(x, \theta)). \tag{22}$$

Since it is a binary classification, there are two output cross-entropies and $L(x, y, \theta)$ represents the average of the cross-entropies. The variable y_i is the provided label, $\hat{y}_i$ is the prediction, x is the input image and θ is the parameters. Usually, an average of all cross-entropies for a chosen number of samples is used as an objective function. Assuming n samples gives a final loss function corresponding to [11]

$$J(\theta) = -\frac{1}{2n} \sum_{j=1}^{n} \sum_{i=1}^{2} y_{j,i} \log(\hat{y}_{j,i}(x, \theta)) + (1 - y_{j,i}) \log(1 - \hat{y}_{j,i}(x, \theta)). \tag{23}$$

A large value of $J(\theta)$ is a consequence of a poor prediction, whereas a small value means that the network predicts well.

In order to derive the binary cross-entropy, the definitions of the two distributions, P and Q, are needed. The distributions are defined by [2]

$$y \in \{0, 1\} \tag{24}$$

$$P(y = 1|x; \theta) = \begin{cases} 1 \\ 0 \end{cases} \tag{25}$$

$$Q(y = 1|x; \theta) = \hat{y}(x, \theta). \tag{26}$$

In Equation (25), $P(y = 1|x; \theta)$ is a conditional probability and is either 0 or 1 depending on which class input x belongs to. By applying the Kullback-Leibler (KL) divergence on the two distributions, $P(x)$ and $Q(x)$, it is possible to derive the cross-entropy. The KL divergence is a measure of the dissimilarity of the two discrete distributions and it is defined by

$$\mathbb{KL}(P||Q) = \mathbb{E}_{X \sim P} \left[\log \frac{P(y|x; \theta)}{Q(y|x; \theta)} \right] \tag{27}$$

$$= \mathbb{E}_{X \sim P} \left[\log P(y|x; \theta) - \log Q(y = 1|x; \theta) \right] \tag{28}$$

$$= \mathbb{E}_{X \sim P} \left[\log P(y|x; \theta) \right] - \mathbb{E}_{X \sim P} \left[\log Q(y|x; \theta) \right] \tag{29}$$

$$= \mathbb{H}(P) - \mathbb{H}(P, Q). \tag{30}$$

In Equation (30), the first term, $\mathbb{H}(P)$, is the system's own entropy and the second term is the cross-entropy, $\mathbb{H}(P, Q)$ [11]. The cross-entropy is thus defined by

$$\mathbb{H}(P, Q) = -\mathbb{E}_{X \sim P} \left[\log Q(y = 1|x; \theta) \right]. \tag{31}$$

Equation (31) is also called the negative log likelihood and is the negative discrete expected value of the logarithm of $Q(y = 1|x; \theta)$, where X is a random variable and P-distributed. By the definition of the discrete expected value, Equation (31) can be rewritten to

$$\mathbb{H}(P, Q) = -\sum_{y} P(y|x; \theta) \log \frac{P(y|x; \theta)}{Q(y|x; \theta)} \tag{32}$$

$$= -\sum_{y} P(y|x; \theta) \log Q(y|x; \theta) + \sum_{y} P(y|x; \theta) \log P(y|x; \theta). \tag{33}$$

The last term in Equation (33) goes to zero, due to the definition of distribution P, and the first term expands to

$$-\sum_{y} P(y|x; \theta) \log Q(y|x; \theta) \tag{34}$$

$$= -P(y = 1|x; \theta) \log Q(y = 1|x; \theta) - P(y = 0|x; \theta) \log Q(y = 0|x; \theta) \tag{35}$$

$$= -y \log \hat{y}(x, \theta) - (1 - y) \log(1 - \hat{y}(x, \theta)). \tag{36}$$

The cross-entropy is completely derived.

2.3.2 Gradient descent variants

There are several algorithms to minimize the objective function and a technique called gradient descent is introduced. Gradient descent is an iterative method where the aim is to achieve the parameters, θ, that minimizes the objective function. By utilize the gradient of the objective function at the current θ, the updated parameters θ' can be obtained. By moving θ in small steps in the opposite direction of the gradient, a small reduction of the objective function is accomplished. The gradient descent updates the parameters according to

$$\theta' = \theta - \alpha \nabla_{\theta} J(\theta), \qquad \text{where} \qquad \alpha > 0. \tag{37}$$

In Equation (37), θ is the current parameters, θ' is the updated parameters, $\nabla_{\theta} J(\theta)$ is the gradient of the objective function, defined by Equation (23), and α is the step size which is a small constant. Each iteration results in one step closer to the local minimum of the objective function. In neural networks, the step size α is the learning rate and is an important parameter with respect to the optimization process.

The gradient descent is a time consuming algorithm, since the gradients of the complete training set or batch size n is calculated. As n increases a single update requires prohibitively

long time [15]. Stochastic gradient descent (SGD), however, calculates the gradient of each sample in the training set. Using batches of the training set and the SGD, resulting in an average and an approximately estimated gradient of the batch and reduction of the variance of the parameters updates, this also leads to more stable convergence. The SGD is in general faster than the ordinary gradient descent. The estimate of the gradient of a batch with n' samples is then defined by

$$\hat{g} = \frac{1}{n'} \nabla_\theta \sum_{j}^{n'} L(x^{(j)}, y^{(j)}, \theta), \tag{38}$$

and the updates of the SGD is described by

$$\theta^{'} = \theta - \alpha \hat{g}. \tag{39}$$

In Equation (39), θ is the current parameters, $\theta^{'}$ is the updated parameters, $\hat{g}$ is the average gradient of the batch and α is the step size which is a small constant. In practice it is common to use higher learning rates in the beginning of the training and later decrease it when approaching the local optimum. The learning rate is decreased due to the introduced noise in the estimation of the gradients. Choosing proper learning rates is a challenge, since small learning rates result in slow convergence and large learning rates can hinder convergence [15].

The SGD algorithm is described by Algorithm 2 and initially the algorithm requires a learning rate schedule and initialization of the parameters. The scheduled learning rates are denoted by α_k at iteration k. Thereafter, an average gradient of the batch, containing the samples $\{x^{(1)}, ..., x^{(n')}\}$, is estimated. Then the average gradient and the corresponding learning rate are used to update the parameters. After the update of the parameters the first iteration is accomplished, it is an iteratively algorithm which stops when the stopping criterion is met.

Algorithm 2 Stochastic gradient descent update [11]

Require: Learning rate schedule α_1, α_2...
Require: Initial parameter θ
$k = 1$
while stopping criterion not met **do**
    Sample a batch of n' examples from the training set $\{x^{(1)}, ..., x^{(n')}\}$
    with corresponding labels $y^{(i)}$.
    Compute gradient estimate: $\hat{g} = \frac{1}{n'} \nabla_\theta \sum_j^{n'} L(x^{(j)}, y^{(j)}, \theta)$
    Apply update: $\theta = \theta - \alpha_k \hat{g}$
    $k = k + 1$
End while

2.3.3 Back-propagation

For each batch the parameters are updated by the SGD and in order to adjust the parameters in all layers, the algorithm back-propagation is required. Back-propagation allows the information from the objective function to flow backward through the network and the gradient

can therefore be estimated for each layer. Back-propagation is thus a method for computing the gradient and the SGD is an algorithm to perform learning from this gradient [11].

Algorithm 3 Back-propagation [11]

After the forward computation, compute the gradient on the output layer:
g $\leftarrow \nabla_{\hat{y}} l(y, \hat{y})$
for $k = l, l-1, ..., 1$ **do**
    Convert the gradient on the layer's output into a gradient into the pre non-linearity activation (element-wise multiplication if f is element-wise):
    g $\leftarrow \nabla_{a^{(k)}} l(y, \hat{y}) = f'(a^{(k)}) \odot g$
    Compute gradients on weights and biases:
    $\nabla_{b^{(k)}} l(y, \hat{y}) = g$
    $\nabla_{W^{(k)}} l(y, \hat{y}) = h^{(k-1)} g$
    Propagate the gradients w.r.t the next lower-level hidden layer's activations:
    $g \leftarrow \nabla_{h^{(k-1)}} l(y, \hat{y}) = W^{(k)} g$
End for

In Algorithm 3 the back-propagation is presented and initiates where the forward propagation finished, i.e. at the gradient of the objective function, $\nabla_{\hat{y}} l(y, \hat{y})$. The number k is the layer and also the iterations of back-propagation. The function f is the activation function, $a^{(k)}$ corresponds to the nodes in layer k, $h^{(k)}$ is the activation vector in layer k and $W^{(k)}$ is the weight matrix. The first iteration in Algorithm 3 is illustrated, using the chain rule of calculus and the first gradient is

$$g \leftarrow \nabla_{a^{(l)}} l(y, \hat{y}) = \left(\frac{\partial \hat{y}}{\partial a^{(l)}} \right)^T \nabla_{\hat{y}} l(y, \hat{y}) = \left(\frac{\partial s(a^{(l)})}{\partial a^{(l)}} \right)^T \nabla_{\hat{y}} l(y, \hat{y}) = f'(a^{(l)})^T g. \tag{40}$$

In Equation (40) the derivative of the activation function and the initial gradient are the element-wise multiplication, which produces the next gradient. The gradient used to update the weights of layer l is defined by

$$\nabla_{W^{(l)}} l(y, \hat{y}) = \left(\frac{\partial f(a^{(l)})}{\partial W^{(l)}} \right)^T \nabla_{\hat{y}} l(y, \hat{y}) \tag{41}$$

$$= \left(\frac{\partial f(W^{(l)} h^{(l-1)} + b^{(l)})}{\partial W^{(l)}} \right)^T \nabla_{\hat{y}} l(y, \hat{y}) \tag{42}$$

$$= h^{(l-1)} f'(W^{(l)} h^{(l-1)} + b^{(l)}) \nabla_{\hat{y}} l(y, \hat{y}) \tag{43}$$

$$= h^{(l-1)} f'(a^{(l)}) \nabla_{\hat{y}} l(y, \hat{y}) = h^{(l-1)} g. \tag{44}$$

The gradient used to update the bias of layer l is described by

$$\nabla_{b^{(l)}} l(y, \hat{y}) = f'(W^{(l)} h^{(l-1)} + b^{(l)}) \nabla_{\hat{y}} l(y, \hat{y}) \tag{45}$$
$$= f'(a^{(l)}) \nabla_{\hat{y}} l(y, \hat{y}) = g. \tag{46}$$

The final step in Algorithm 3 is defined by

$$g \leftarrow \nabla_{h^{l-1}} l(y, \hat{y}) = \frac{\partial \hat{y}}{\partial h^{(l-1)}} \nabla_{\hat{y}} l(y, \hat{y}) = \frac{\partial f(W^{(l)} h^{(l-1)} + b^{(l)})}{\partial h^{(l-1)}} \nabla_{\hat{y}} l(y, \hat{y}) \tag{47}$$
$$= W^{(l)} f'(a^{(l)}) \nabla_{\hat{y}} l(y, \hat{y}) = W^{(l)} g. \tag{48}$$

By applying these kinds of calculations it is possible to propagate the gradients backwards through the network until the first layer is reached.

2.3.4 Momentum and Nesterov momentum

Around a local optimum is it common that the surface curves are steeper in one dimension than in another, and for the SGD it is difficult to navigate these so called ravines. The SGD oscillates across the slopes of the ravine and hesitantly approaches the local optimum. Momentum is a method that speeds up the SGD in the relevant direction and reduces the oscillations. This is achieved by taking both the steps of the current gradient and the last update step into account, as described by

$$v_t = \gamma v_{t-1} + \alpha \nabla_\theta J(\theta), \quad \gamma \in [0, 1) \tag{49}$$
$$\theta = \theta - v_t. \tag{50}$$

Where v is called the velocity vector and γ is the momentum term, which is usually recommended to be set to 0.9 [15]. Moreover, v is called velocity since it is the direction and speed of the parameters through the parameter space. If γ is set to zero, the ordinary SGD update is obtained, thus γ adjusts how much the momentum impacts the update. If all gradients point in the same direction, the momentum term will increase for these dimensions. Therefore the updates for dimensions whose gradients change directions are reduced, resulting in a reduction of the oscillations and faster convergence [15]. For example, the gradient descent can be seen as a ball rolling down on the loss surface and the momentum adds mass into the system allowing the ball to roll over small local bumps.

The Nesterov momentum is described by

$$v_t = \gamma v_{t-1} + \alpha \nabla_\theta J(\theta - \gamma v_{t-1}) \tag{51}$$
$$\theta = \theta - v_t. \tag{52}$$

Where also here the γ is the momentum term and normally set to a value around 0.9. The term $\theta - \gamma v_{t-1}$ is an approximation of the next position of the parameters. In other words, it

is a prediction of where the future parameters are located. The gradient with respect to the approximate future position of the parameters are thus calculated, instead of the gradient with respect to the current parameters. As a reference to the ball, the ball in this case is not rolling down a hill blindly following the slope, instead the ball knows to slow down before the slope reaches the bottom [15].

2.3.5 Optimizers

There are several modern methods based on SGD where the learning rate is adaptive, meaning that the learning rates depend on the properties or the magnitude of the gradients [11]. In this thesis the RMSProp and ADAM algorithm are introduced.

The algorithm RMSProp is a short for root mean squared propagation and adapts the learning rate with respect to the past gradients. It uses a moving average of the squared gradients for each weight to normalize the gradient used to update the parameters. This results in an effect of balancing the learning rate, i.e. for a small gradient the learning rate increases to avoid vanishing and large gradient the learning rate decreases to avoid divergence. The moving average of the squared average is also the second moment of the gradients, or the mean, and is defined by

$$r_t = \rho r_{t-1} + (1-\rho) g_t \odot g_t. \tag{53}$$

Where ρ is the decay rate, r_{t-1} the previous moving average and g_t the gradient at iteration t. In Equation (53), the moving average of the gradients depends only on the previous average and the current gradient. To update the parameters, the gradients are divided by the square root of the moving average of its recent magnitude. Instead of a global scalar learning rate, a vector of learning rates for each parameter is used. The parameter update is described by

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\delta + r_t}} \odot g_t. \tag{54}$$

The updated parameters are the θ_{t+1}, θ_t is the current parameters, α is the learning rates, δ is a small constant for numerical stabilization, r_t is the moving average defined by Equation (53) and g_t the current gradient.

The RMSProp algorithm is presented in Algorithm 4 and initially it requires a global learning rate α, decay rate ρ, initial parameters θ, small constant δ for stabilize division by small numbers and sampled batch size with corresponding labels. The moving average, r, of the squared gradients is initialized as zero. If the decay rate is $\rho = 0.9$, then in the first iteration of RMSProp the running average is defined by

$$r = (1-\rho) g \odot g = 0.1 g \odot g. \tag{55}$$

Thus, r is used to update the parameters and g is the computed average gradient of the objective function. In order to obtain the parameter update, the learning rate α element-wise multiplied with the current gradient is divided by the square root of r, see Equation (54), and thereafter the first iteration can be completed. In the next iterations, r is a sum of

gradients which are recursively defined as a decaying average of all past squared gradients, and is used to update the parameters. This is performed iteratively until the stopping criterion is met.

Algorithm 4 RMSProp [11]

Require: Global learning rate α, decay rate ρ
Require: Small constant δ used for stabilize division by small numbers
Require: Initial parameters θ
Initialize accumulation variables $r = 0$
Initialize time step $t = 0$
while stopping criterion not met **do**
    Sample a batch of n' examples from the training set $\{x^{(1)}, ..., x^{(n')}\}$ with
    corresponding labels $y^{(i)}$
    Compute gradient: $g_t \leftarrow \frac{1}{n'} \nabla_\theta \sum_{i=1}^{n'} L(x^{(i)}, y^{(i)}, \theta)$
    Accumulate squared gradient: $r_t \leftarrow \rho r_{t-1} + (1 - \rho) g_t \odot g_t$
    Compute parameter update: $\Delta\theta = -\frac{\alpha}{\sqrt{\delta + r_t}} \odot g_t$ ($\frac{\alpha}{\sqrt{\delta + r_t}}$ applied element-wise)
    Apply update $\theta_{t+1} \leftarrow \theta_t + \Delta\theta$
end while

The decay rate ρ is commonly set to 0.9 and the default learning rate is recommended to be $\alpha = 0.001$ [15].

ADAM is a short for adaptive moment estimation and is just as the RMSProp an algorithm with adaptive learning rates for each parameter. The learning rates are derived from estimates of the first and second moment of the gradients. The first and second moment correspond to the mean and variance respectively [15] and are defined by

$$s_t = \rho_1 s_{t-1} + (1 - \rho_1) g_t \tag{56}$$
$$r_t = \rho_2 r_{t-1} + (1 - \rho_2) g_t \odot g_t. \tag{57}$$

Equation (56) is the first moment of the gradients, which include the decay rate ρ_1 for the first moment estimate, s_{t-1} is the previous first moment and g_t is the gradient. In Equation (57), which is the second moment of the gradients, ρ_2 is the decay rate for the second moment estimate, r_{t-1} is the previous second moment and g_t is the gradient.

The RMSProp updates are computed with the second moment of the gradients, whereas ADAM updates also take advantage of the first moment of the gradients. In ADAM algorithm, the moving averages are estimates of first and second order momentum of the gradient, and there are therefore two decay rates. ADAM also has the bias-correction term which is important for the convergence, because without correcting the bias will result in large step sizes and this can cause divergence [13]. Unlike in ADAM, the second moment estimate in RMSProp may have a high bias early in training. This is due to the fact that the RMSProp does not include the bias-correction [11]. The bias-corrections are defined as

$$\hat{s} = \frac{s}{1 - \rho_1^t} \tag{58}$$

$$\hat{r} = \frac{r}{1 - \rho_2^t}. \tag{59}$$

Where $\hat{s}$ and $\hat{r}$ are the correct bias in first moment and second moment respectively. Furthermore, s and r are the first and second moments. The decay rates are ρ_1 and ρ_2, which are raised to the power of t.

The parameter update for ADAM is thus defined by

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{s}_t}{\sqrt{\hat{r}_t} + \delta}. \tag{60}$$

Where the learning rates are α, θ_{t+1} is the updates parameters, θ_t is the current parameters, δ is a small constant for numerical stabilization, $\hat{s}_t$ and $\hat{r}_t$ are the correct biases in first and second moment respectively.

The ADAM algorithm is described by Algorithm 5, and it requires a global learning rate α, first and second moment s and r, respectively, initialized as zero, the constants ρ_1 and ρ_2 (hyper-parameters that control the exponential decay rates of the first and second moment), small constant δ for numerical stabilization, initial parameters θ and a sampled batch with the corresponding labels before it can start iterating. After the initializing step, the gradient is used to obtain the first and second moment estimates, defined by Equation (56) and 57. First and second moment estimates are thereafter utilized to correct the biases according to Equation (58) and 59. Finally, the parameters can be updated by using the calculation according to Equation (60). Just as RMSProp, ADAM algorithm is an iteratively method and therefore the parameters are updated until the stopping criterion is met.

Algorithm 5 ADAM [11]

Require: Global learning rate, α
Require: Decay rates for moment estimates, $\rho_1, \rho_2 \in [0, 1)$
Require: Small constant δ used for numerical stabilization
Require: Initial parameters θ
Initialize 1st and 2nd moment variables, $s = 0$, $r = 0$
Initialize time step $t = 0$
while stopping criterion not met **do**
    Sample a batch of n' examples from the training set $\{x^{(1)}, ..., x^{(n')}\}$ with
    corresponding labels $y^{(i)}$.
    Compute gradient $g_t \leftarrow \frac{1}{n'} \nabla_\theta \sum_{i=1}^{n'} L(x^{(i)}, y^{(i)}, \theta)$
    Update biased first moment estimate: $s_t \leftarrow \rho_1 s_{t-1} + (1 - \rho_1) g_t$
    Update biased second moment estimate: $r_t \leftarrow \rho_2 r_{t-1} + (1 - \rho_2) g_t \odot g_t$
    Correct bias in first moment: $\hat{s}_t \leftarrow \frac{s_t}{1-\rho_1^t}$
    Correct bias in second moment: $\hat{r}_t \leftarrow \frac{r_t}{1-\rho_2^t}$
    Compute update: $\Delta\theta = -\alpha \frac{\hat{s}_t}{\sqrt{\hat{r}_t} + \delta}$
    Apply update: $\theta_{t+1} \leftarrow \theta_t + \Delta\theta$
end while

The ρ_1 and ρ_2 are recommend to be set to 0.9 and 0.999 respectively. The default learning rate $\alpha = 0.001$ and small constant $\delta = 10^{-8}$ are common default settings [13].

2.3.6 Regularization

Over-fitting is a problem that arises in machine learning, it means that the parameters are too adjusted to the training data and performs poorly on test data. There are regularization strategies that are implemented in order to avoid over-fitting. In this thesis, early stopping and dropout are introduced.

The main idea behind early stopping is to stop the training before the networks over-fits. This is done by comparing the training loss and the validation loss, if the training loss is decreasing and the validation loss is increasing after a number of epochs the network is over-fitting. Every time the validation loss is improved, the model is stored. The training stops when no parameters have improved after the best recorded validation loss for some number of iterations, which is set in advance. This is a popular regularization technique since it is simple and effective, as well as it does not damage the learning dynamics. This is due to the fact that there is almost no change in the underlying training procedure, the objective function or the set of parameter values. On the other hand, the early stopping requires a validation set and this means that the training data decreases [11].

The concept of the dropout technique is to randomly drop neurons between the different layers, which offer a way to approximately combining many different architectures. By dropping a neuron means that the neuron is temporarily removed from the network, therefore is the dropout technique computationally inexpensive and powerful. The dropout rate, usually

denoted as p, is set in advance and a fixed probability of dropping the neuron. The dropout technique is implemented only during training and not during testing [16].

3 Method

The methodology of this thesis is divided in two parts, which are image preprocessing and classification. Image preprocessing is an essential part of an image recognition algorithm, in regards to performance, and the purpose is to achieve a structure of the input data that is advantageous for the learning. The classification algorithm is based on a CNN, where the aim is to distinguish tumor tissue from normal tissue in images in three dimensions. Since the images of the lung tissue are in three dimensions, 3D CNNs are used. Two different 3D CNNs are designed to classify images of lung tissue in three dimensions, using a varying number of layers and different hyper-parameter settings.

In order to investigate a simpler case, an artificial data set representing lungs in two dimensions is used. A 2D image classifier is thus implemented using the generated artificial data set. In a 2D CNN it is more convenient to follow the input image through the network than for a 3D CNN, due to the fact that a 2D image is more comprehensible. The different operations in the 2D CNN can thus be plotted. This is performed to obtain an overview of how the different layers in a CNN are operating on the image.

In this chapter, the data set, the image preprocessing, the different architectures and the hyper-parameters are presented. The architectures are the two 3D CNNs for the images containing lung tissue and the 2D CNN for the artificial data set. Different evaluation metrics needed are also discussed in order to evaluate the architectures.

3.1 Data set and materials

Annotated data is required in order to train a network, meaning that each image is provided a label. The data set analyzed in this thesis consist of CT scans, which are examined and diagnosed by three radiologists [9].

3.1.1 Data set

The data set used is the LUNA16 [9] and consist of 888 CT scans, where 601 scans contain cancer tissue and 287 scans do not contain cancer tissue. Each scan consists of different number of slices and the number of slices per scan can vary between 100 and 400, where each slice has the dimension 512×512 pixels.

3.1.2 Annotation

There are two associated annotation tables to the data set. The first table contains information about positions (x, y, z) and diameters of all tumors in the data set. In the second table, positions of potential tumors are provided along with a classification. The potential tumors represents both tumors and normal lung tissue, where a tumor is annotated with a 1 and normal tissue with a 0. The first mentioned table contains 1186 tumors and the other table contains 551 065 potential tumors, where the 1186 tumors from the first table are the same in the second table.

3.2 Image preprocessing

The aim of the preprocessing of the data set is to facilitate the training and the learning of the CNN. The learning can improve by manipulating the pixel values of the image into smaller numbers with less variation, which is achieved using interpolation, normalization and zero centering.

The images are expressed in Hounsfield units (HU) that is a quantity scale for radio-density. For example, water and air correspond to 0 HU and -1000 HU respectively, whereas in the body can HU reach 3000 for bone.

3.2.1 Interpolation

The data set consists of CT scans produced by different machines, leading to a differing spacing between the voxels in the images. The images are re-sampled using interpolation in order to achieve an uniform spacing. By introducing a new spacing, i.e. $[1, 1, 1]$, a resize factor can be derived and utilized to interpolate new data points.

The voxels are rearranged and the coordinates of the tumor from the annotation table need to be converted, due to the interpolation. The coordinates are converted using the new spacing and origin of the CT scan, where the origin is the coordinates of the voxel with index $(0, 0, 0)$ in physical units. The voxel coordinates are calculated by

$$\hat{c} = \frac{|c - u|}{r}, \tag{61}$$

where the $\hat{c}$ is the the new position, c is the original position, u is the origin and r is the wanted spacing [18].

3.2.2 Normalization and zero centering

In HU, the number -1000 corresponds to air and values greater than 400 corresponds to bones, where the intermediate values represent tissues. Since the aim is to differentiate tumor tissue from normal tissue this is the pixel range of interest. The pixel values in the CT scans are in the range of -1024 to 2000, indicating that there are unessential pixel values that can be eliminated.

Normalization and zero centering are important for the training, since updates of the gradients converge more efficient if the crucial features have pixel values in a similar range with less variation. It can be challenging for the weight sharing in a CNN if the pixel values in the image varies in a wide range.

In order to normalize the pixel values, n_{min} and n_{max} are introduced and set to -1000 and 400 respectively. The constants n_{min} and n_{max} are utilized to transform the pixel values in the range of -1000 to 400 HU into a number in the range of 0 to 1. The normalization is defined by

$$I_{norm} = \frac{I - n_{min}}{n_{max} - n_{min}}. \tag{62}$$

Where I_{norm} is the normalized image and I is the input image. The normalization leads to that the pixel values greater than 400 HU assuming a number greater than one and those values are set to one in order to achieve numbers in the range of 0 to 1. With similar reasoning, the pixel values less than -1000 normalizes into a negative number and are thus set to zero.

The images are thereafter zero centered, meaning that the pixel mean of the complete data set is subtracted from the normalized pixel values. The pixel mean used is 0.25 [18] and the purpose is to zero center the normalized image.

3.3 Preparation of data set

The essential difficulties with this data set are the variation of dimension and the large size of the CT scans. This is due to the CNN that requires the images to have consistent dimension and the restriction of memory and computational capacity of the computer. The CT scans consist of a too large amount of data being used as input for the CNN, instead smaller cubes can be extracted and used for training.

3.3.1 Extraction of preprocessed images

Since the annotation tables provide coordinates of tumors and normal tissues, parts of the preprocessed CT scans can be utilized to obtain a data set that consists of images with consistent dimension. Volumes that encircle the coordinates can be extracted from the preprocessed CT scans, generating a new data set. From the CT scans two data sets are generated, containing images of dimension $16 \times 16 \times 16$ respectively $32 \times 32 \times 32$. The data set containing images of dimension $16 \times 16 \times 16$ is only used to efficiently investigate the performance using different settings of the network, since the cubes do not encircle large tumors. Both data sets consist of 1186 images containing tumors and 1186 images containing normal tissue.

The volumes are extracted so that the provided coordinates are in the center of the volume. For a few cases the coordinates are too close to the boundary of the image, resulting in that the cube to be extracted is moved in the correct direction. Consequently, the tumors are still in the volume but not in the center.

3.3.2 Data set distribution

The extracted data sets are divided in three subsets, i.e. training set, validation set and test set which is seen in Table 1. This distribution of the data set is commonly used in deep learning. The training set is used during training to adjust the parameters. The validation set is also used during training to validate the performance, but no adjustments of the parameters are based on the validation set. After the training the test set is introduced to evaluate the network.

All data sets are randomly shuffled and each sample has an associated label. The label is $[1, 0]$ or $[0, 1]$ depending on if it is a tumor or non-tumor respectively.

	Data set	Training set	Validation set	Test set
Proportion	100%	70%	20%	10%
Number of samples	2372	1680	468	224

Table 1: Data set distribution.

3.4 Classification

Three architectures are discussed in this section. Initially, the 2D CNN for the artificial data set that is built of two convolutional layers. The others are the two 3D CNNs that are represented as Architecture 1 and Architecture 2, consisting of three and six convolutional layers respectively. Both architectures are using the two data sets, i.e. the one containing images of dimension $16 \times 16 \times 16$ and the other containing images of dimension $32 \times 32 \times 32$. The images of the smaller dimension are used to efficiently investigate hyper-parameters, number of layers and the two optimizers. The two optimizers used are the RMSProp and ADAM, discussed in Section 2.3.5.

3.4.1 Network layers

The architectures consist of several layers with different operations, which include convolutional layers, max pooling layers, flatten layer and dense layer. In the architectures, each convolutional layer is followed by a ReLu activation function, max pooling layer and during training a dropout layer. After the final max pooling layer a flatten layer is applied, resulting in a vector of dimension $N \times 1$, where N is the number of output elements in the final max pooling layer. In other words, the flatten layer reduces the dimensions of the output of the max pooling layer into a vector. The binary output is achieved by applying the dense layer, which is a fully connected layer, that reduces the output of the flatten layer into a binary output.

The output of the dense layer is transformed to a probability by using the softmax function, described in Section 2.1.3. With this probability it is possible to conclude what class the image belongs to.

3.4.2 Classification of artificial data

The artificial data set and the binary 2D CNN are discussed in this section. The data set consists of 2D images representing lungs containing one or several tumors, but there are also images not containing tumors.

The images are randomly generated by varying the size of the different features and shades of gray, see Figure 8. The generated images have the dimension 128×128.

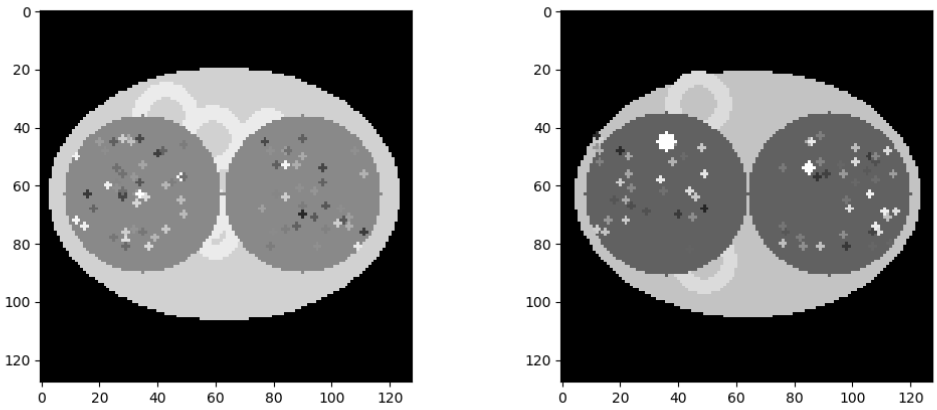

Figure 8: Two example images of the artificial data set, representing a healthy patients lungs respectively a patient diagnosed with cancer. The white larger marks in the image to the right are representing tumors.

The artificial images are preprocessed by resizing the dimension of the images to 64×64 and dividing the pixel values by 255, since it is the largest pixel value the images are normalized. The training set consists of 800 generated images, 228 images for the validation set and 114 for the test set.

The generated artificial data set is used to train the 2D CNN presented in Table 2, which consists of two convolutional layers and follows the description in Section 3.4.1. In Table 2 the kernel size of the layers is introduces and for instance, the dimension of the output of the first convolutional layer is $64 \times 64 \times 4$, meaning that there are four feature maps of dimension 64×64.

Layer	Kernel size	Activation	Output
Input			64 × 64 × 1
Conv1	3×3	ReLu	64×64×4
MaxPool	2×2		32×32×4
Conv2	3×3	ReLu	32×32×8
MaxPool	2×2		16×16×8
Flatten			1×2048
Dense		Softmax	1×2

Table 2: Architecture of the 2D CNN.

The ADAM optimizer is used in order to train the 2D CNN, where the learning rate is equal to 0.001. The other hyper-parameters are set as described in Section 3.4.5, except the dropout rate that is set to $p = 0.4$.

Both images in Figure 9 are introduced to the network after the training process. The images are nearly identical, where the difference is the larger white mark in the image to the right. The aim is to compare two nearly identical images by plotting them after the operating layers, leading to following the images through the operations and observing what features the network is responding to.

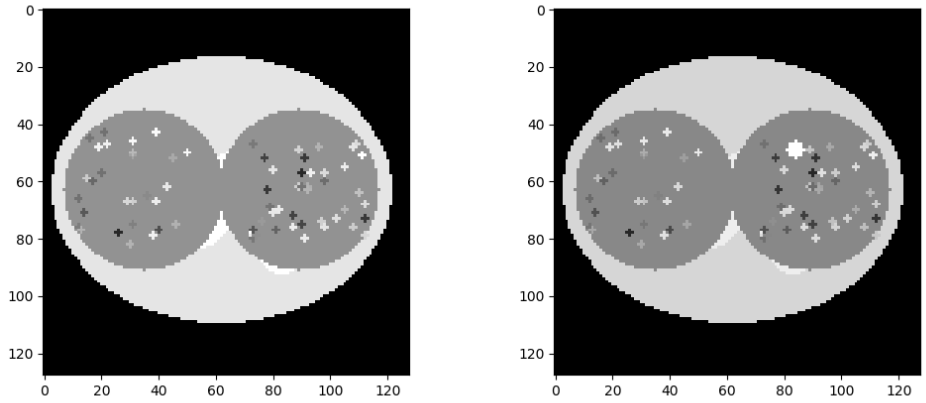

Figure 9: Two nearly identical images of the artificial data set. The difference is that the image to the right is containing a tumor, representing the larger white mark.

3.4.3 Classification of tumors using Architecture 1

Architecture 1 is designed to classify images of lung tissue and consist of three convolutional layers in three dimensions. The two data sets used are the images of lung tissue of the dimension $16 \times 16 \times 16$ respectively $32 \times 32 \times 32$. The architecture is presented in Table 3, along with the kernel size of the layers and the activation function. Since two different data sets have been used there are two columns for the output in Table 3, referred as to Output 1 and Output 2 respectively. The hyper-parameters and other settings are discussed in Section 3.4.5.

3.4.4 Classification of tumors using Architecture 2

Architecture 2 is just as for Architecture 1, described in Section 3.4.3, designed to classify the two data sets of images of lung tissue. The architecture consists of six convolutional layers and is inspired by Chon et al. 2017 [6], which is presented in Table 4 and the structure of the table is equivalent to Table 3.

In Table 4 there is one less convolutional and max pooling layer for the images of dimension $16 \times 16 \times 16$, in comparison to using images of dimension $32 \times 32 \times 32$. This is because of a dimensional error at the fifth max pooling layer, where the feature map cannot be reduced.

Layer	Kernel size	Activation	Output 1	Output 2
Input			16×16×16×1	32×32×32×1
Conv1	3×3×3	ReLu	16×16×16×8	32×32×32×8
MaxPool	2×2×2		8×8×8×8	16×16×16×8
Conv2	3×3×3	ReLu	8×8×8×16	16×16×16×16
MaxPool	2×2×2		4×4×4×16	8×8×8×16
Conv3	3×3×3	ReLu	4×4×4×32	8×8×8×32
MaxPool	2×2×2		2×2×2×32	4×4×4×32
Flatten			1×256	1×2048
Dense		Softmax	1×2	1×2

Table 3: The architecture of Architecture 1.

Layer	Kernel size	Activation	Output 1	Output 2
Input			16×16×16×1	32×32×32×1
Conv1	3×3×3	ReLu	16×16×16×8	32×32×32×8
MaxPool	2×2×2		8×8×8×8	16×16×16×8
Conv2	3×3×3	ReLu	8×8×8×16	16×16×16×16
MaxPool	2×2×2		4×4×4×16	8×8×8×16
Conv3	3×3×3	ReLu	4×4×4×32	8×8×8×32
MaxPool	2×2×2		2×2×2×32	4×4×4×32
Conv4	3×3×3	ReLu	2×2×2×64	4×4×4×64
MaxPool	2×2×2		1×1×1×64	2×2×2×64
Conv5	3×3×3	ReLu	1×1×1×128	2×2×2×128
MaxPool	2×2×2		-	1×1×1×128
Conv6	3×3×3	ReLu	-	1×1×1×256
Flatten			1×128	1×256
Dense		Softmax	1×2	1×2

Table 4: The architecture of Architecture 2.

3.4.5 Hyper-parameters

The hyper-parameters for the CNNs are discussed in this section. Hyper-parameters are the parameters that are defined before training, unlike the parameters that are adjusted during training. In this thesis, variations of hyper-parameters such as the learning rate, two different optimizers, number of layers and feature maps are investigated.

Batch size
The batch size is set to 64 and 20 for the images of dimension $16 \times 16 \times 16$ and $32 \times 32 \times 32$ respectively.

Parameter initialization
The parameter initialization is important for the convergence and cannot be defined arbitrarily. The weights are therefore initialized using a uniform initializer, where the weights are drawn from the uniform distribution defined by

$$W^{(k)} \sim U\left(-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}\right). \tag{63}$$

Where n_{in} and n_{out} are the number of input and output weights, respectively, in the weight tensor. The bias is initialized as zero [10].

Stride length and kernel size
The stride length for the convolutional and max pooling layer is set to 1. The dimension of the kernels in the convolutional layers in 2D and 3D is set to 3×3 and $3 \times 3 \times 3$ respectively, whereas the kernels of the max pooling layers are set to 2×2 and $2 \times 2 \times 2$ respectively.

Zero padding
The convolutional operation results in a reduction of the input image or feature map, which can be avoided by adding zeros to the boundary and the technique is called zero padding. For example, if an input image has length $W_{input} = 32$ in any direction, kernel size $F = 3$, stride length $S = 1$ and zero padding $P = 1$, then

$$W_{output} = \frac{W_{input} - F + 2P}{S} + 1 = \frac{32 - 3 + 2 \cdot 1}{1} + 1 = 32. \tag{64}$$

The output W_{output} is the length of the feature map. Furthermore, $P = 1$ implies that one line of zeros is added at each boundary of the image [14].

Optimizers
For ADAM and RMSProp there are hyper-parameters that are set to the default values. In ADAM the default settings are $\rho_1 = 0.9$, $\rho_2 = 0.999$ and $\delta = 10^{-8}$. In RMSProp the default settings are $\rho = 0.9$ and $\delta = 10^{-8}$ [10]. Several learning rates are examined for both optimizers to find an appropriate learning rate for this mission.

Regularization
Dropout and early stopping are implemented in order to avoid over-fitting. The dropout

rate is set to $p = 0.2$, i.e. the proportion of the input nodes to drop. With early stopping another hyper-parameter called patience appear, which is the number of epochs after the early stopping and is set to 5.

3.5 Evaluation metrics

When the training of a network is completed it needs to be analyzed with respect to performance. There are several evaluation metrics that can be utilized to analyze the performance, such as accuracy, values of the objective function, sensitivity, specificity, confusion matrix and receiver operating characteristic curve.

The accuracy is a measure of the ability of a network to predict the inputs and is defined by Equation (65), where all the correct predictions are summed and divided by the total number of predictions. Depending on the value of the prediction it is rounded to zero or one and compared to the true label. If the prediction and the label are equal, then it is a correct prediction [10]. Normally, the average accuracy is plotted as a function of epochs and the plot provides an overview of the average accuracy during the training.

The average loss is usually plotted as a function of epochs, which corresponds to the values of the objective function. This plot provides an overview of the minimization of the objective function. It is common to have validation loss and training loss in the same plot, and if the validation loss is increasing and training loss is decreasing it is an implication of over-fitting.

The sensitivity is a measure of how good the network is at predicting a tumor, and is the probability that a prediction will be positive when a tumor is present. It is also called the true positive rate. The specificity is a measure of how good the network is at classifying non-tumors, and is the probability that a prediction will be negative when a tumor is not present. The specificity is also called the true negative rate. The sensitivity and the specificity are defined by Equation (66) and Equation (67) respectively.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{65}$$

$$Sensitivity = \frac{TP}{TP + FN} \tag{66}$$

$$Specificity = \frac{TN}{TN + FP}. \tag{67}$$

Where TP is true positive, the number of correct classifications of the tumors. The TN is true negative, the number of correct classification of the non-tumor. The FP is false positive, the number of the tumors that have been classified as non-tumors. The FN is false negative, the number of the non-tumors that have been classified as tumors [5].

Normally, TP, TN, FP and FN are used to obtain a confusion matrix, defined by Equation (68). The higher values on the diagonal, the better classifier.

$$Confusion \quad matrix = \begin{pmatrix} TP & FN \\ FP & TN \end{pmatrix} \tag{68}$$

The so called receiver operating characteristic curve (ROC curve) is a plot that is considered. The ROC curve illustrates the sensitivity as a function of false positive rate, defined by $1 - Specificity$. The false positive rate is sometimes called the probability of false alarm.

The ROC curve is a plot of different cut-off points, where each point on the ROC curve represents a sensitivity/specificity pair. The sensitivity/specificity pair corresponds to a certain classification threshold and by using several classification thresholds, several sensitivity/specificity pairs can be utilized in order to plot the ROC curve.

The ROC curve illustrates the predictability of a CNN. For example, if the threshold is $0.1, 0.2, .., 0.9$, the TP, FN, TN and FP can be calculated. Thereafter, it is possible to obtain the true positive rate and true false positive rate for each threshold and the ROC curve can be plotted.

The sensitivity and specificity are equal 1.0 if the algorithm classifies the test data perfectly, resulting in that the area under the ROC curve (AUC) is equal to one. The AUC is thus a measure of how well the algorithm classifies the input data.

4 Result

In this chapter, the results of the different architectures are presented. The results of the 2D CNN using the artificial data set is discussed in Section 4.1. Then the results of Architecture 1 and Architecture 2 are introduced in Section 4.2 and 4.3 respectively. In these two sections there is a section containing the results of different optimizers and learning rates using images of dimension $16 \times 16 \times 16$ and a section where the result is based on the images of dimension $32 \times 32 \times 32$. Lastly Section 4.4, presenting the resulting statistics of Architecture 1 of 100 trainings.

4.1 Artificial data set

In order to achieve an architecture that perform well, different number of layers and feature maps have been investigated. The outstanding architecture is presented in Table 2. By using the optimizer ADAM with learning rate 0.001, the accuracies in Table 5 were obtained.

Learning rate	Training acc.	Validation acc.	Test acc.	Epochs
0.001	99.3%	96.9%	97.4%	97

Table 5: The resulting accuracies of the training, validation and test set using ADAM optimizer.

In Figure 10, the plot to the left illustrates the average accuracy as a function of epochs, whereas the plot to the right represent the average loss as a function of epochs. Both plots contain results of both training and validation set, and provide an overview of how the network is performing and learning.

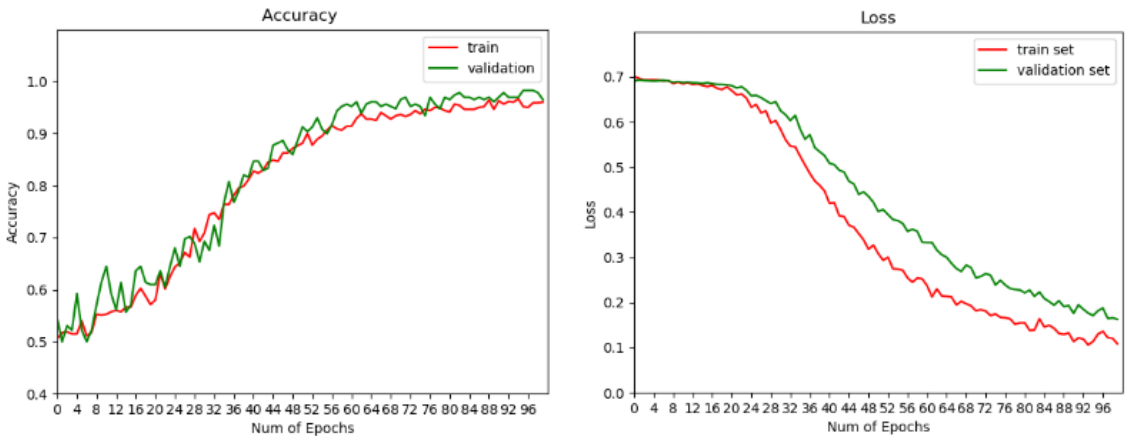

Figure 10: The two plots illustrates the average accuracy and loss as a function of epochs.

After training, the feature maps can be plotted by using the images in Figure 9 as inputs, and the only difference of the images is that one contains a tumor. The feature maps are shown in Figure 11, 13 and 12, 14, by comparing them it is possible to observe how the 2D CNN responds to the two input images differently. In the latter mentioned figures the

representing tumor is there, whereas in the first mentioned figures it is not. This shows how the CNN is trained to detect tumors. In Figure 11, the resulting feature maps of the first convolutional layer with the activation function applied are illustrated, using the image not containing a tumor. Figure 12 presents the feature maps of the final max pooling layer, i.e. the layer before the flatten layer.

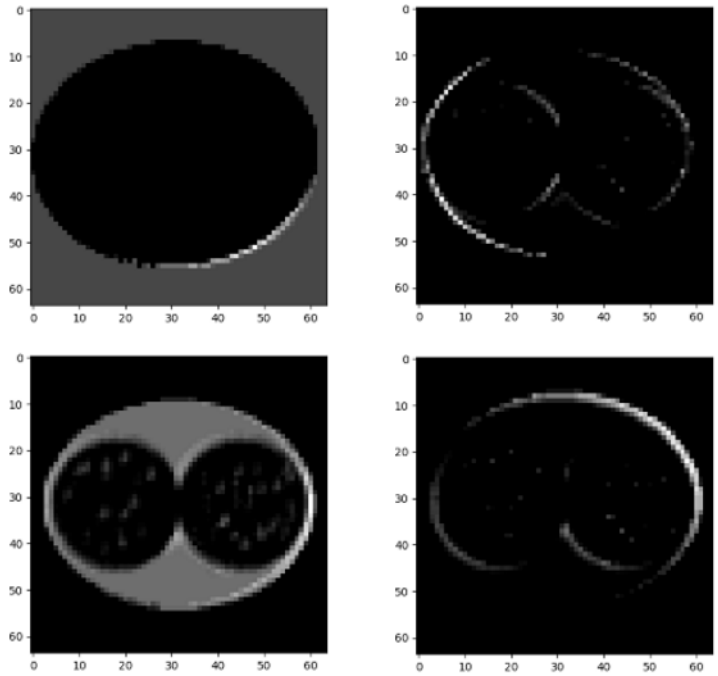

Figure 11: Illustration of the feature maps of the first convolutional layer, where the activation function is applied. The input image is the image to the left in Figure 9, i.e. the one not containing a tumor.

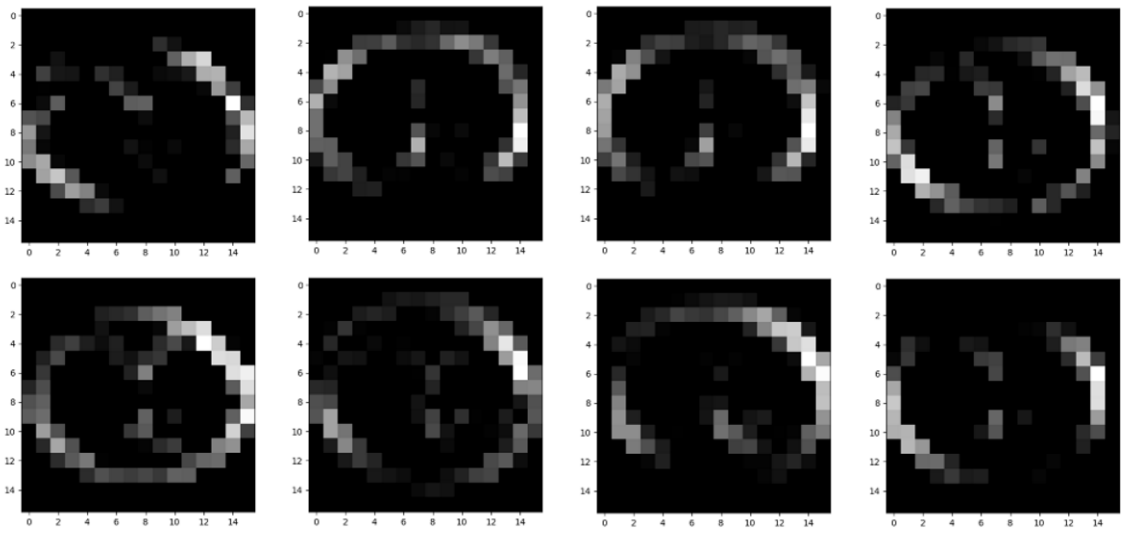

Figure 12: Illustration of the feature maps of the last max pooling layer. The input image is the image to the left in Figure 9, i.e. the one not containing a tumor.

Figure 13 and 14, represent the feature maps of the first convolutional layer, with the activation function applied, and the final max pooling layer respectively. The input image used contains a tumor.

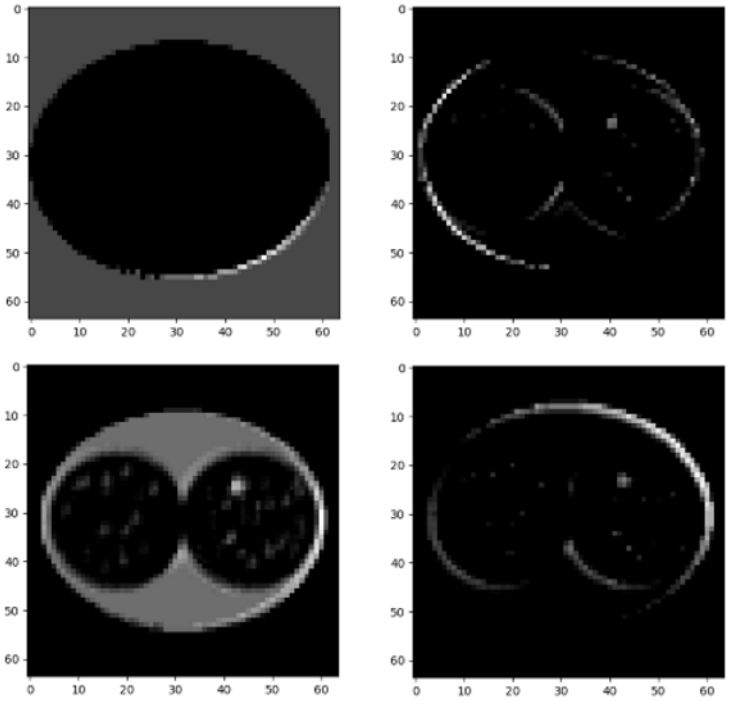

Figure 13: Illustration of the feature maps of the first convolutional layer, where the activation function is applied. The input image is the image to the right in Figure 9, i.e. the one containing a tumor.

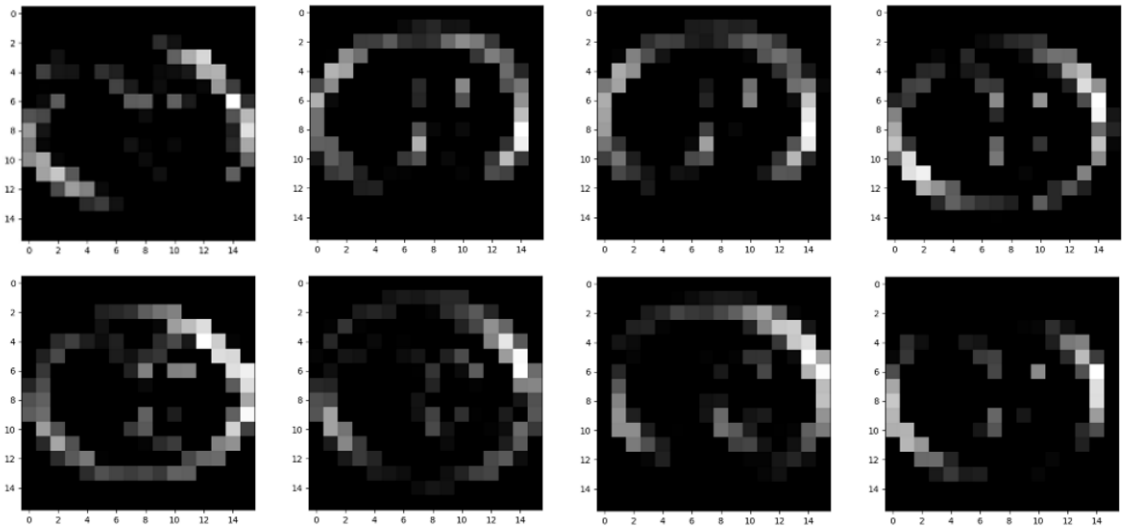

Figure 14: Illustration of the feature maps of the last max pooling layer. The input image is the image to the right in Figure 9, i.e. the one containing a tumor.

4.2 Classification of tumors using Architecture 1

Architecture 1 is built of three convolutional layers followed by an activation function and a max pooling layer. In Section 4.2.1, the result of two different optimizers with three different learning rates is presented, using images of dimension $16 \times 16 \times 16$.

In Section 4.2.2, the result of Architecture 1 using images of dimension $32 \times 32 \times 32$, ADAM optimizer and a learning rate of 0.001 is discussed.

4.2.1 Optimizers and learning rates

Initially, the results of using ADAM optimizer with the three learning rates 0.01, 0.001 and 0.0001 are discussed. Afterwards, the results of the RMSProp optimizer using the equivalent learning rates are presented.

The results of using ADAM optimizer are presented in Table 6, i.e. the accuracies of the training, validation and the test set, number of epochs used and time consumed of three trainings with different learning rates.

Learning rate	Training acc.	Validation acc.	Test acc.	Epochs	Time (min)
0.01	90.5%	85.3%	88.8%	10	8.3
0.001	99.8%	92.5%	92.4%	21	17.4
0.0001	96.4%	92.1%	90.2%	42	34.9

Table 6: The result of using ADAM optimizer with three different learning rates and images of dimension $16 \times 16 \times 16$

In Figure 15 three plots of the average loss as a function of epochs, for both training and validation set, with three different learning rates are illustrated.

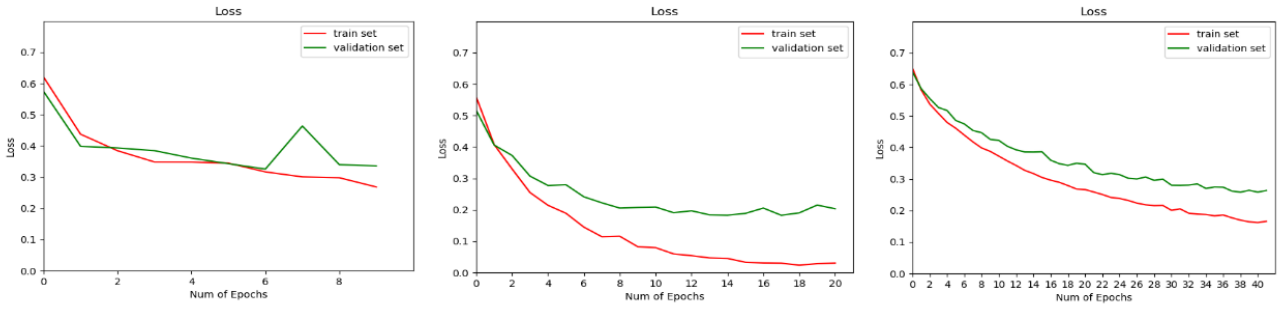

Figure 15: The three plots present how the loss is decreasing during training using ADAM optimizer, for both training and validation set. The three different learning rates are 10^{-2}, 10^{-3} and 10^{-4} respectively.

The second optimizer, RMSProp, is also implemented for three different learning rates using images of dimension $16 \times 16 \times 16$. The accuracies for the training, validation and the test set,

number of epochs used and time consumed of three trainings with different learning rates are presented in Table 7.

Learning rate	Training acc.	Validation acc.	Test acc.	Epochs	Time (min)
0.01	93.2%	89.7%	91.5%	10	8.5
0.001	99.8%	94.0%	95.5%	19	16.4
0.0001	98.8%	92.1%	93.8%	94	77.8

Table 7: The result of using RMSProp optimizer with three different learning rates and images of dimension $16 \times 16 \times 16$.

In Figure 16, three plots containing the average loss as a function of epochs for the three learning rates are illustrated. The plots present the results of both training and validation set.

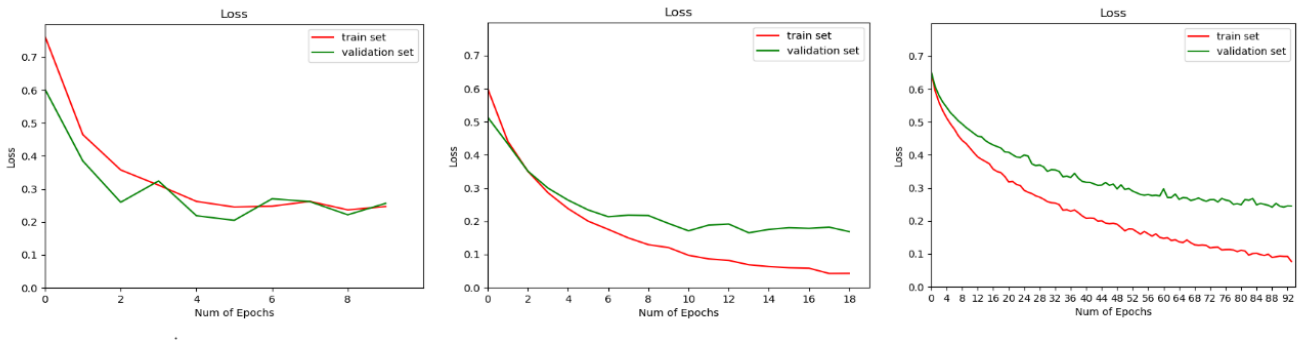

Figure 16: Three plots of the decreasing loss during training using RMSProp optimizer, for both training and validation set, with three learning rates. The three different learning rates are 10^{-2}, 10^{-3} and 10^{-4} respectively.

4.2.2 Classification

In this section, the ADAM optimizer with learning rate 0.001 is implemented using images of dimension $32 \times 32 \times 32$.

In Table 8, the accuracies for the training, validation and the test are presented, as well as the number of epochs used and the time consumed of the training.

Learning rate	Training acc.	Validation acc.	Test acc.	Epochs	Time (min)
0.001	99.4%	90.8%	91.1%	23	137.3

Table 8: The result of using ADAM optimizer with learning rate 0.001 and images of dimension $32 \times 32 \times 32$.

The average accuracy and loss as functions of epochs of the training and validation set are illustrated in Figure 17.

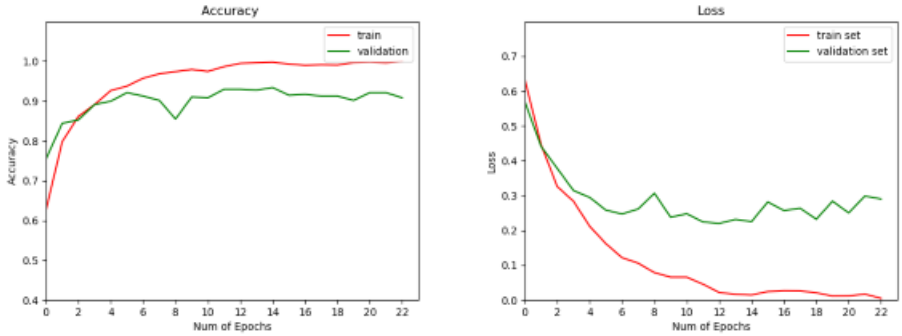

Figure 17: Plots of the average accuracy and loss as functions of epochs, where ADAM optimizer with learning rate 0.001 is implemented. The plots contain result of both training and validation set.

Figure 18 consists of two plots, which are the confusion matrix and ROC curve of the test set.

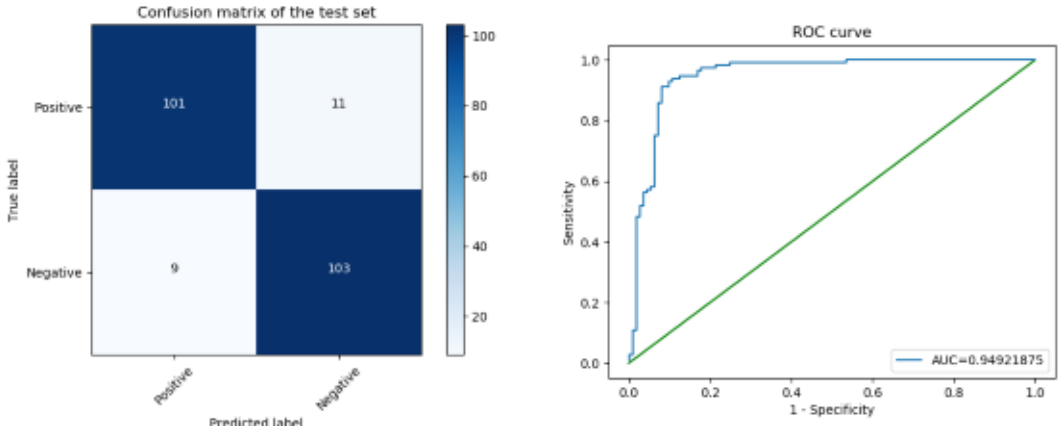

Figure 18: The confusion matrix is to the left and the ROC curve is to the right, both are obtained by using the test set. By using the values in the confusion matrix, the sensitivity and specificity is calculated to 90.2% and 92.0% respectively. The AUC value of the ROC is 0.95.

4.3 Classification of tumors using Architecture 2

Architecture 2 consists of six convolutional layers, followed by an activation function and a max pooling layer. In Section 4.3.1, the result of two different optimizers with three different learning rates is presented. Architecture 2 is trained using images of dimension $16 \times 16 \times 16$.

In Section 4.3.2, Architecture 2 is trained using images of dimension $32 \times 32 \times 32$, ADAM optimizer and a learning rate of 0.001.

4.3.1 Optimizers and learning rates

The results of using ADAM optimizer with the three learning rates 0.01, 0.001 and 0.0001 are discussed. Afterwards, the results of the RMSProp optimizer with the equivalent learning

rates are presented.

In Table 9, the results of ADAM optimizer with the three learning rates are presented. The results include the training accuracy, validation accuracy, test accuracy, number of epochs used and training time.

Learning rate	Training acc.	Validation acc.	Test acc.	Epochs	Time (min)
0.01	50.0%	50.0%	50.0%	15	15.8
0.001	99.8%	94.9%	92.0%	35	26.8
0.0001	98.4%	90.4%	90.2%	100	76.7

Table 9: The result of using ADAM optimizer with three different learning rates and images of dimension $16 \times 16 \times 16$.

In Figure 19, three plots of the average loss as a function of epochs with different learning rates, for both training and validation set, are illustrated.

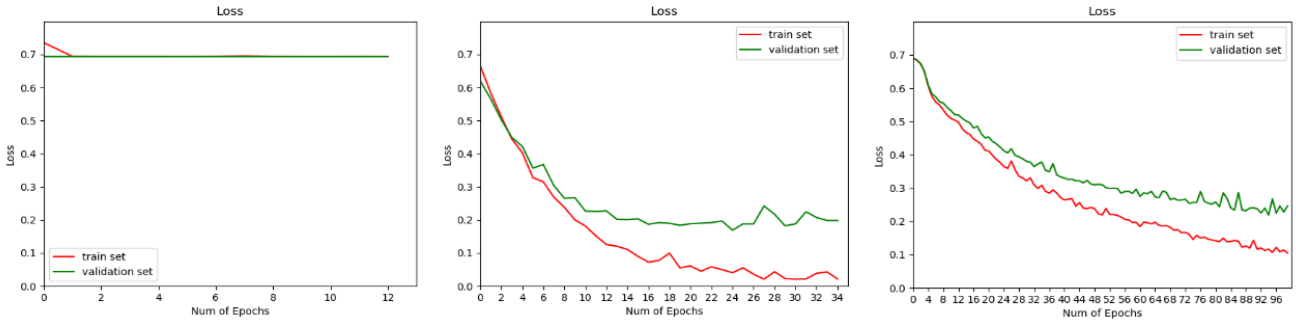

Figure 19: Three plots of the decreasing loss during training using ADAM optimizer, for both training and validation set, with different learning rates. The three different learning rates are 10^{-2}, 10^{-3} and 10^{-4} respectively.

The RMSProp optimizer is implemented for three learning rates and the results presented in Table 10 are the accuracies for the training, validation and test set, number of epochs used and time consumed of three trainings using different learning rates.

Learning rate	Training acc.	Validation acc.	Test acc.	Epochs	Time (min)
0.01	97.7%	90.0%	90.6%	40	30.0
0.001	97.3%	91.5%	88.0%	46	37.5
0.0001	97.3%	90.6%	94.6%	100	78.4

Table 10: The result of using RMSProp optimizer with three different learning rates and images of dimension $16 \times 16 \times 16$.

Figure 20 presents three plots of the average loss as a function of epochs, for both training and validation set, with the different learning rates.

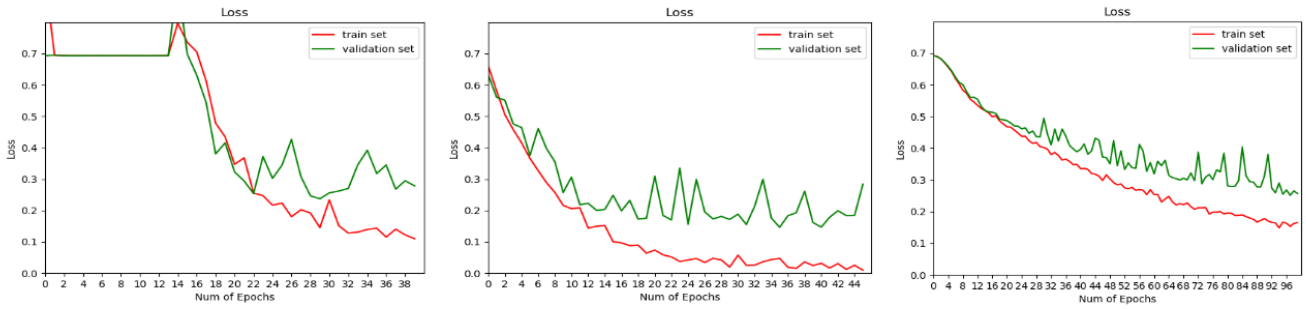

Figure 20: The three plots of the decreasing loss during training using RMS-Prop optimizer, for both training set and validation set. The three different learning rates used are 10^{-2}, 10^{-3} and 10^{-4} respectively.

4.3.2 Classification

As in Section 4.2.2, ADAM optimizer with learning rate 0.001 is implemented using images of dimension $32 \times 32 \times 32$.

The result of a training is presented in Table 11.

Learning rate	Training acc.	Validation acc.	Test acc.	Epochs	Time (min)
0.001	100.0%	91.2%	92.4%	25	156.0

Table 11: The result of using ADAM optimizer with learning rate 0.001 and the dimension of the images is $32 \times 32 \times 32$.

The average accuracy and loss as functions of epochs, for the training and validation set are illustrated in Figure 21.

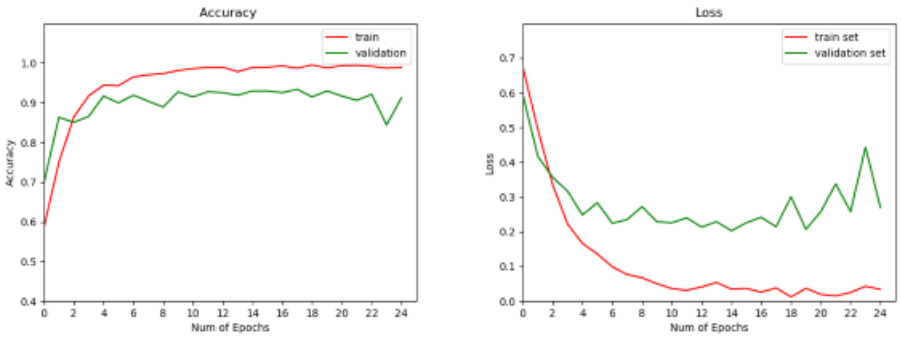

Figure 21: Plots of the average accuracy and loss as functions of epochs, where ADAM optimizer with learning rate 0.001 is implemented. The result of both training and validation set is presented.

Figure 18 consists of the confusion matrix and the ROC curve, obtained by using the test set.

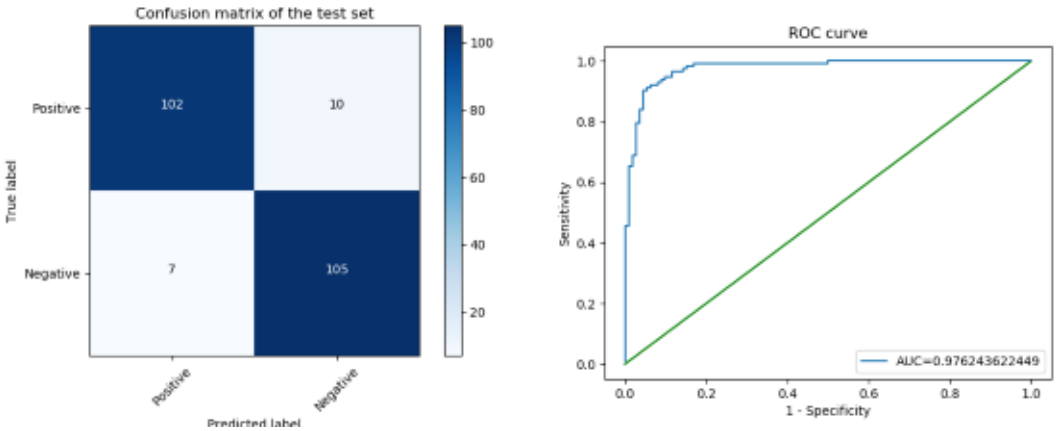

Figure 22: The confusion matrix is the left and the ROC curve is to the right, where both are obtained by using the test set. By using the values in the confusion matrix, the sensitivity and specificity is calculated to 91.1% and 93.8% respectively. The AUC value of the ROC is 0.98.

4.4 Statistics

The results of different trainings vary, even though the architecture and the parameter settings are consistent. Architecture 2 is therefore trained 100 times in order to obtain mean values and standard-deviations. The optimizer ADAM with learning rate 0.001 is implemented using images of dimension $32 \times 32 \times 32$.

The different mean values of the accuracy, sensitivity and specificity for the three data sets are presented in Table 12.

	Training set	Validation set	Test set
Accuracy	99.8%	90.9%	91.1 %
Sensitivity	99.8 %	89.8 %	89.7 %
Specificity	99.8 %	92.1 %	92.4%

Table 12: The mean values of 100 trainings using ADAM optimizer and learning rate 10^{-3}.

The standard-deviations of the accuracy, sensitivity and specificity of the three data sets are presented in Table 13.

	Training set	Validation set	Test set
Accuracy	0.32	2.04	2.39
Sensitivity	0.01	0.03	0.04
Specificity	0.003	0.04	0.05

Table 13: The standard-deviations of 100 trainings using ADAM optimizer and learning rate 10^{-3}.

Figure 23 illustrates a histogram of the accuracies of the test data during 100 trainings.

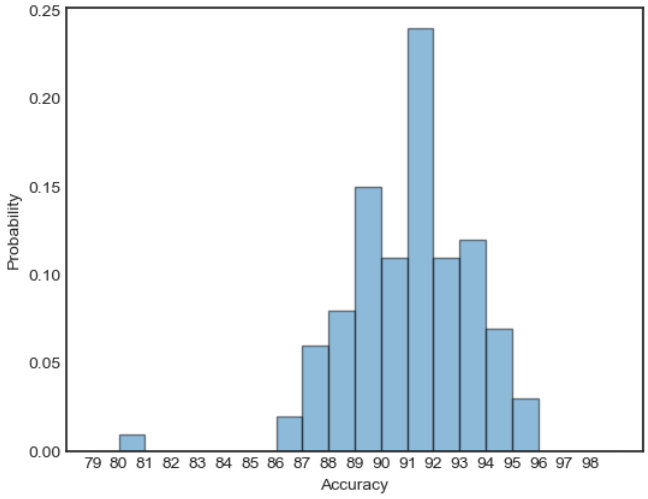

Figure 23: A histogram of the test accuracies of 100 trainings.

5 Discussion and Conclusion

In this chapter the result is discussed and conclusions are drawn. In Section 5.1 the results of the 2D CNN trained on the artificial data set are discussed. Thereafter the discussion of the results of Architecture 1 and 2 using the images of lung tissue are presented in Section 5.2. Finally, the discussion about future work and application of the algorithm are done in Section 5.3 and 5.4 respectively.

5.1 Artificial data set

In order to find a suitable and well performing CNN, different architectures were examined. Initially, the number of feature maps in each layer was half of the feature maps in the architecture described in Table 2. After training this architecture, the input images were plotted for different operations and layers in the CNN, resulting in images with vague features. The feature maps were therefore duplicated twice, with respect to the first implemented architecture, resulting in several feature maps that did not vary significantly and the optimal architecture is thus described by Table 2.

By observing Figure 11 - 12 and Figure 14 - 13 it is possible to follow the propagating input images and understand how the layers are operating on the images in a CNN. Figure 14 - 13, also illustrates that the CNN is responding to the tumor, since the images are containing the tumor in both the first and last layer. This was the purpose of generating an artificial data set and training a network with it. It means that the CNN is recognizing the tumors and generalizing new data. The plots in Figure 10, also imply that the CNN is learning during training, since both the loss values and the accuracies are improving.

5.2 Classification of lung tumors

Initially in this section, the hyper-parameters, settings and the impact on the performance are discussed. Thereafter the two architectures are compared and the performance is discussed.

5.2.1 Hyper-parameters and settings

There are many parameters to consider building a CNN, with an infinite number of combinations. In this thesis, the parameters and settings to determine have been restricted to the learning rate, two different optimizers, dropout rate, number of feature maps and number of layers.

Determining a learning rate is a trade-off between time and an accurate model. It is more likely to approach the minimum with a small learning rate, but it results in more epochs and time consumption. Whereas a large learning rate is leading to approaching of the minimum with less epochs, but can instead miss the minimum. The learning rate was determined by comparing the loss plots in Figure 15 and 16 for Architecture 1, as well as Figure 19 and 20 for Architecture 2, and the most suitable learning rate for both architectures and optimizers is 0.001. This conclusion is based on training time, the loss value and its behavior in the

loss plots.

By observing the plots in Figure 15 and 16 the performance of ADAM optimizer is slightly better. The loss plot of the validation set indicates that the ADAM optimizer achieves the lowest loss values. The dropout rate was also investigated, and the dropout rate $p = 0.2$ is leaving a stable loss plot for the validation set. If the dropout rate was set to zero a large variation in the loss plot of the validation set emerged, whereas a large dropout rate resulted in a time consuming training. A large dropout rate results in time consuming training, since units dropped are too many and the architectures vary too much during training.

There is not a significantly difference between the performance of the two architectures, but Architecture 1 requires less epochs and time for a training. The loss plots for Architecture 1 are also more smooth and do not vary as much as for Architecture 2, which occur when there are too many feature maps and the network is over-fitting. With this result we can conclude that more layers do not necessarily improve the performance, which was not expected with these complex images.

5.2.2 Performance

According to both AUC values, presented in Figure 18 and 22, the architectures perform well, since the AUC values are close to 1.0. In Figure 18 and 22 both the CNNs have a higher specificity than sensitivity, which means that the network is a better predictor of normal tissue. In the health care, high sensitivity is preferable since it is crucial for a patient to be diagnosed with cancer if it is present. The result of the 100 trainings of architecture indicates that Architecture 1 is performing well, with high average accuracy, sensitivity and specificity.

It was unexpected that such a simple and shallow network, as Architecture 1, could perform that well. This can be derived from the image preprocessing and the extracted volumes. The manipulated pixel values were adequate for the network to learn from and the features of the images were possible to distinguish. The extracted volumes leads to that the tumor tissue is a greater proportion of the images, in comparison with the tumor tissue in a complete CT scan, and therefore the features of the tumor tissues are more distinguishable.

5.3 Future work

In this thesis the tumors have been extracted from the CT scans since the locations are provided, but in reality when the patient is examined by a radiologist the location of a tumor is not known. A future work could therefore be to create an algorithm that detects potential tumors, a so called region of interest. Since the data set used in this thesis contains images with located tumors it is possible to train a two dimensional CNN that uses each slice of the CT scan as an input and detects suspicious regions. When the coordinate of a region is obtained it is possible to extract a volume and feed the three dimensional CNN with it. If you had unlimited computational capacity it would be possible to use the whole CT scan as an input to a 3D CNN, then finding regions of interests in a 2D image would not be necessary.

The CNN in this thesis classifies the images to have a tumor or not, i.e. it classifies abnormalities in the input image. A future work could also be developing an algorithm that classifies the tumor to be benign or malignant. In order to do this development, annotation about the malignancy is required.

In order to improve the results, the images containing tumors could be grouped into several data sets, consisting of images containing tumors with diameters in a similar range. By using these data sets the network could be trained separately to recognize tumors with a diameter in a certain range. The features of the tumors will be more similar and therefore it is reasonable that the results can be improved. Another improvement could be to extend the algorithm so that more information than just images is taken into account in order to classify whether a patient has cancer or not. Parameters that can be considered are e.g. age, smoker or non-smoker.

5.4 Application

Since both sensitivity and specificity are high for the developed algorithm, it could be implemented as a good predictor of both tumor and normal tissue in the health care. The algorithm is classifying extracted cubes of the CT scan, and it could be applied so that a radiologist can select a region of interest by pointing at a touch screen showing the slices of a CT scan. Thereafter a volume encircling the region of interest can be extracted and used as input for a three dimensional CNN. This application enables classification of images containing potential tumors. Another application is to divide the CT scan into several cubes, and use all of them as input for the classification algorithm. With this application it is possible to use the complete CT scan as input, which can result in reduced effort and time for the radiologist in comparison with the previous mentioned application. The algorithm could thus be put into effect as decision support for the radiologists, resulting in saving time and providing a second opinion.

References

[1] *CS231n Convolutional Neural Networks for Visual Recognition.* Standford University. http://cs231n.github.io/linear-classify/, 2018-02-28.

[2] *ECE521 Lectures 9 Fully Connected Neural Networks.* University of Toronto. http://www.psi.toronto.edu/~jimmy/ece521/Lec9-nn2.pdf, 2018-01-20.

[3] *Dödlighet i lungcancer.* Folkhälsomyndigheten, 2018. https://www.folkhalsomyndigheten.se/folkhalsorapportering-statistik/folkhalsans-utveckling/halsa/dodlighet-i-cancer/lungcancer-dodlighet/, 2018-04-05.

[4] *Lungcancer.* Cancerfonden, 2018. https://www.cancerfonden.se/om-cancer/lungcancer, 2018-04-10.

[5] Byung Bok Ahn. *The Compact 3D Convolutional Neural Network for Medical Images.* Standford University. 2017.

[6] Niranjan Balachandar Albert Chon and Peter Lu. *Deep Convolutional Neural Networks for Lung Cancer Detection.* Standford University. 2017.

[7] Geoffrey E. Hinton Alex Krizhevsky, Ilya Sutskever. *ImageNet Classification with Deep Convolutional Neural Networks.* https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf, 2017-12-12.

[8] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer Science + Business Media, LLC, 2006.

[9] LUNA16 Grand Challenge. https://luna16.grand-challenge.org/,2018-01-10.

[10] François Chollet et al. Keras. https://github.com/keras-team/keras, 2015.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. http://www.deeplearningbook.org, 2017-12-11.

[12] Anna Gummeson. Prostate cancer classification using convolutional neural networks. *Master's Theses in Mathematical Sciences*, 2016.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[14] Micheal A. Nielsen. *Neural Networks and Deep Learning.* Determination Press, 2015. http://neuralnetworksanddeeplearning.com/chap6.html, 2017-12-11.

[15] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[17] Sun-Chong Wang. Artificial neural network. In *Interdisciplinary computing in java programming*, pages 81–100. Springer, 2003.

[18] Guido Zuidhof. *Full Preprocessing Tutorial*. `https://www.kaggle.com/gzuidhof/full-preprocessing-tutorial`, 2018-01-10.

TRITA -SCI-GRU 2018:248