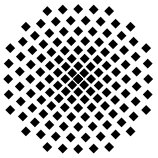


University of Stuttgart
Institute for Signal Processing and System Theory
Professor Dr.-Ing. B. Yang

Research Project S1230

Prostate Cancer Detection in MR-images with `Keras` Deep Learning library

Prostata Krebs Erkennung in MR-Bildern mit `Keras` Deep Learning

Author: Larissa T. Triess

Date of work begin: 2017/11/01
Date of submission: 2018/02/28

Supervisor: Prof. Dr.-Ing. Bin Yang
Prof. Dr. Itsuo Kumazawa
Annika Liebgott, M.Sc

Keywords: Machine learning, `Keras`, medical image processing, cancer detection, binary classification

Prostate cancer is the most common cancer related cause of death in men. Most often, it can be treated successfully when diagnosed, making early detection important. With magnetic resonance imaging (MRI) many kinds of diseases can be detected. However, scans are difficult to interpret, leading to inconsistency in diagnosis. A computer-aided detection can help to increase agreement among doctors. This work focuses on classifying MR-scans into cancer and healthy cases with the help of a neural network. It achieved a classification accuracy of up to 97%.

Contents

1. Introduction

Prostate cancer is the second most common cancer in men and the most common cancer related cause of death. Most often, it can be treated successfully when diagnosed, making early detection important. Magnetic resonance imaging (MRI) is a powerful, noninvasive imaging tool, capable of accurately detecting many kinds of diseases. It has advantage over transrectal ultrasound guided biopsy, as it does not require several invasive operations, thus can be applied to a larger patient population. MRI is the most accurate imaging method for prostate cancer detection, but it requires the expertise of experienced radiologists to make the correct diagnosis. As there exists a vast variety of experience and inconsistency across doctors, a computer-aided detection can increase agreement among doctors.

This work focuses on reliable classification of MR-scans into cancer and healthy cases to assist doctors in making a diagnosis. A simple neural network (NN) is used, implemented with the `Keras` deep learning library. The dataset used for training contains 3460 images of 128 healthy patients, and 1465 images of 218 patients with prostate cancer. Most scans are of size 521x521 pixels. They enter a preprocessing stage before training in order to crop them and normalize the pixel values to a range of [0, 1). The initial network architecture is composed of a stack of three convolutional layers with a Rectifier Linear Unit (ReLU) activation and max-pooling layer, followed by two fully-connected layers. For each image input, it outputs a probability vector. It states the classification probability per class, i.e. cancer and healthy cases.

Many parameters influence the performance of such a classifier. Several of these hyper-parameters and input settings are tested and discussed in this work. The initial setting achieved an accuracy of about 60% on the test set, leaving much space for improvement. A significant challenge is the phenomenon of heavy overfitting. In most cases, the training accuracy reached almost 100% only after a few epochs, but the validation accuracy remained low. This means the model does not generalize well. However, it was possible to improve the accuracy by 37% from the initial setting to reach up to 97% on the test set by tuning the network and input parameters.

The following gives a short overview of the structure of this work. Chapter 2 introduces other related work about this topic. In chapter 3 the fundamentals of machine learning, NNs, and their components are described. The development environment, details about the dataset and the implementation are contained in chapter 4. The results are presented in chapter 5. It is subdivided into the experiment setup and our baseline value, six experiment sections, and a summary section. The experiments feature investigation of several different parameter and input data settings, such as optimizer type, learning rate, class weights, data augmentation and more. In the end, chapter 6 summarizes the whole work and gives an outlook on possible future work.

2. Related Work

Prostate cancer is the second most common cancer in men (after skin cancer) and the most common cancer related cause of death [1]. About 19% of cancers in men occur in the prostate, and approximately 16% of all men will be diagnosed with prostate cancer in their lifetime (28% in 2013) [2].

Figure 2.1 shows the cancer statistics of the United States of America for both men and women from 1975 to 2013.

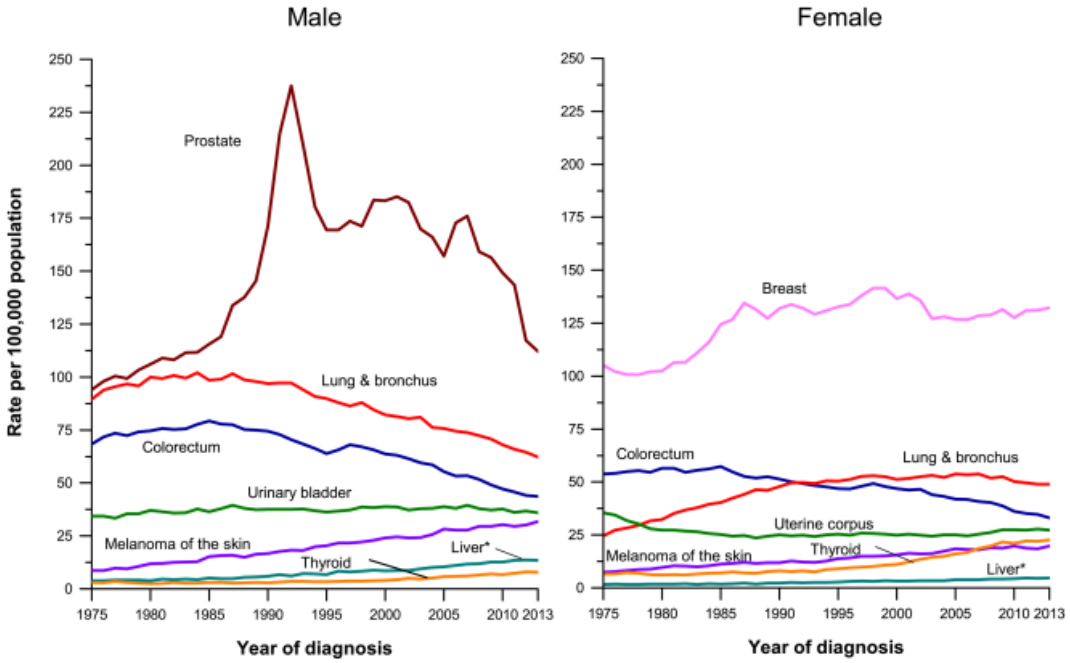


Figure 2.1.: Trends in incidence rates for selected cancers by sex in the USA from 1975 to 2013. Rates are age adjusted to the 2000 US standard population and adjusted for delays in reporting (source: [2]).

Prostate cancer often can be treated successfully when diagnosed, therefore a preferably early detection is important. Currently, the most used clinical method is random systematic (sextant) biopsy under the guidance of transrectal ultrasound (TRUS) if a patient suffers from a high prostate-specific antigen (PSA) level or has an abnormal screening digital rectal examination (DRE). TRUS-guided biopsy for prostate cancer detection has high false-negative results, thus biopsy is usually repeated 5-7 times to increase the detection accuracy. As it is an invasive operation, it is not suitable for screening a large patient population. MRI on the other hand is a powerful, noninvasive imaging tool. The latter causes an important advantage for MRI over the biopsy method. It is able to accurately detect or diagnose many kinds of diseases by providing anatomical, functional and metabolic MRI information. An explanation of MRI is given in appendix A.2.

The state-of-the-art diagnostic method uses MR imaging with transrectal ultrasound-guided biopsy. Multiparametric MRI (mpMRI) images, e.g. T2-weighted imaging, diffusion-weighted imaging (DWI), and apparent diffusion coefficient (ADC) MR images are very beneficial in targeting biopsies towards suspicious cancer regions and can help select subjects who should undergo anterior biopsy. This can lead to a higher detection accuracy. MRI is the most accurate imaging method for prostate cancer detection (PCD) but it requires the expertise of experienced radiologists. The variety of experience across doctors causes inconsistency in diagnoses, therefor a computer-aided detection (CAD) can be beneficial to increase agreement among doctors.

In general, there are five main stages in a typical CAD system for PCD: (a) image preprocessing, (b) segmentation, (c) registration, (d) feature extraction, (e) classification. Preprocessing of the data is needed to normalize and transform the raw images to a domain where cancer regions can be detected easier. It is better to only analyze the prostate region, therefore in segmentation the prostate region usually is segmented from the whole MR image for subsequent analysis. Registration transforms different sets of images onto one coordinate system and feature extraction derives distinctive information/characteristics from from targets of interest in the whole prostate region. Finally, classification identifies whether an unknown region/volume belongs to cancer region or not, based on the extracted features.

In the last decade, researchers have proposed many CAD systems for PCD using different types of features and classification methods. Chan et al. [3] first applied magnetic resonance (MR) images for PCD in 2003. Two types of second-order features, i.e. texture features extracted from lesion candidate and anatomical features described by cylindrical coordinate, and a support vector machine (SVM) were employed to identify prostate cancer in the peripheral zone (PZ). Appendix A.1 includes a short description of the different prostate zones.

Niaf et al. [4] trained and compared four different classifiers: nonlinear SVM, linear discriminant analysis, k-nearest neighbors and naive Bayes classifiers. In their method, suspicious cancer regions of interest (ROIs) within the PZ are manually annotated by both histopathologist and researcher, and then classified into benign and cancer tissues. They use a feature set including first-order statistics, Haralick features [5], gradient features, semi-quantative and quantative dynamic parameters extracted from T2-weighted, DWI and dynamic contrast-enhanced (DCE) MR images. Fig. 2.2 shows a work-flow of a typical prostate CAD system which includes these image types.

In 2015, Kwak et al. proposed a method using texture features based on Local Binary Patterns (LBPs) [7]. Their dataset consists of three categories: MR-positive prostate cancers, benign MR-positive lesions, and MR-negative benign lesions. With a three-stage feature selection, an area under receiver operating characteristic curve (AUC) of 0.89 was achieved for distinguishing between cancer and MR-positive or MR-negative benign lesions.

Most existing PCD methods focus on identifying prostate cancer in the PZ, instead of the entire prostate region. In order to detect cancer regions from the whole prostate, usually a sufficient number of suspicious ROIs need to be provided by a histopathologist. This is very time consuming and sensitive to manual errors. Therefore, Qian et al. [8] proposed a framework to directly identify the prostate cancer regions from in vivo MRI by using random forests and an auto-context model to effectively integrate features from multi-source images. They regarded each voxel in prostate regions as ROI and directly detected the location of the

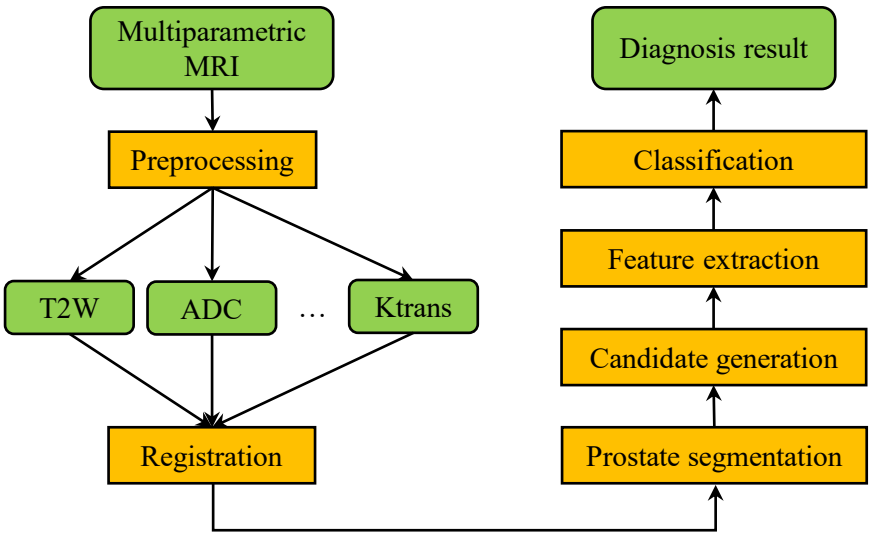


Figure 2.2.: Work-flow of a typical prostate CAD system. Green rectangles indicate data (original scans and images after preprocessing), yellow rectangles indicate processes applied to the data or images (source: [6])

prostate cancer from the whole region, where random Haar-like features were used for both appearance and context features.

Due to the recent success in convolutional neural networks (CNNs) many different approaches of CNN-based prostate detection have been published lately. The main promise is to replace handcrafted features with multiple levels of representations of data in an unsupervised manner. Deep learning provides promising results in many applications, such as computer vision or speech processing. Furthermore some studies have shown deep learning-based methods gain much better performance than methods using handcrafted or low-level features [9, 10].

Tsehay et al. proposed a prostate cancer recognition architecture in 2017 [11] which is based on the "Hollistically nested Edge Detection (HED)" of S. Xie and Z. Tu [12]. HED takes a whole image as input and produces a probability map image. For cancer detection Tsehay et al. used a dataset of 52 patients with T2-weighted (T2W), ADC, and B2000 MR images which were combined to a three channel RGB image (c.f. figure 2.3). The network architecture has five side outputs from five different convolutional layers with a different receptive field each. The side outputs are combined to a fused output with which a detection rate of 86% and a false positive rate of 20% was obtained.

In this work we face a few restrictions concerning the resources compared to some of the papers mentioned above. For once, our dataset only consists of one MR image type, unlike e.g. [11], so no three channel RGB image can be generated. Furthermore, it is not possible to generate 3D representations of the prostate, even though for some patients several scans exist, i.e. several layers of the prostate. Due to incompleteness and varying number of scans per patient, we focus on 2D data in this work. Section 4.2 gives more information about the dataset. As the given time frame to conduct this work is very limited, it focuses on classifying whether an MR scan contains a cancer region or not, not the localization of the tumor. Furthermore, we want to detect the cancer in the entire prostate region, not only in the PZ, which means we are dependent on large amount of labeled data. The limitation here

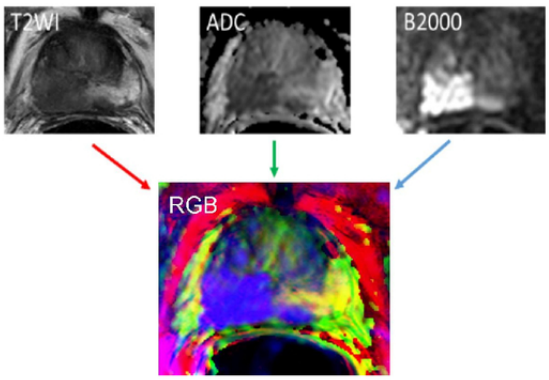


Figure 2.3.: Generating a three channel RGB image from three mpMRIs (source: [11])

is, that the dataset does not provide prostate segmentation maps, which means the location of the prostate has to be approximated. The main goal is to find a result as good as possible with a small and simple neural network and the restricted resources given.

3. Basic Principles

This chapter gives a short overview on machine learning and the most important concepts. For more information have a look at David Kriesels book "Ein kleiner Überblick über neuronale Netze" [13] or the `Coursera` course "Machine Learning" by Stanford University [14]. An introduction to the anatomy of the prostate and the MR-imaging technology is given in appendix A.1 and A.2, respectively.

3.1. Introduction to Machine Learning

Machine learning is a subfield of artificial intelligence (AI) that aims to understand the structure of data and fit it into models, that can be used and understood by humans. This section gives an overview on the historical development of machine learning and explains the fundamentals of neural networks and machine learning.

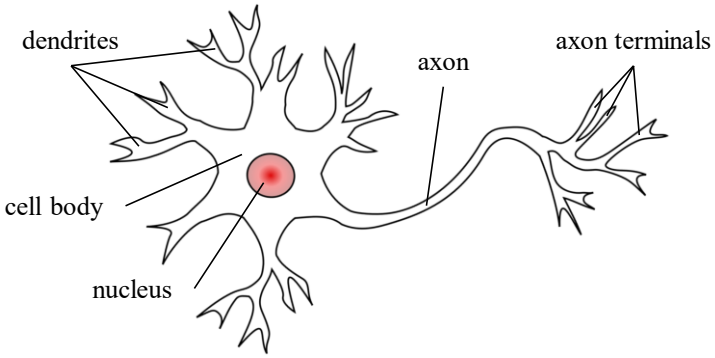
3.1.1. History

Though we are currently living in a boom of machine learning, it is neither a new approach nor has it been popular all the time. After F. Rosenblatt designs the perceptron in 1958 [15] A. Samuel first mentions the term "machine learning" in [16]. Due to bad progress in this field, not much attention was payed to machine learning the following years. In the 1980s a temporary new interest in machine learning arose. The availability of big amount of data in the 2000s revived the interest in machine learning, while the term "Deep Learning" was first mentioned in 2006 by G. Hinton. By today many businesses apply machine learning techniques and new markets are being created, such as personalized advertisement or new health care structures.

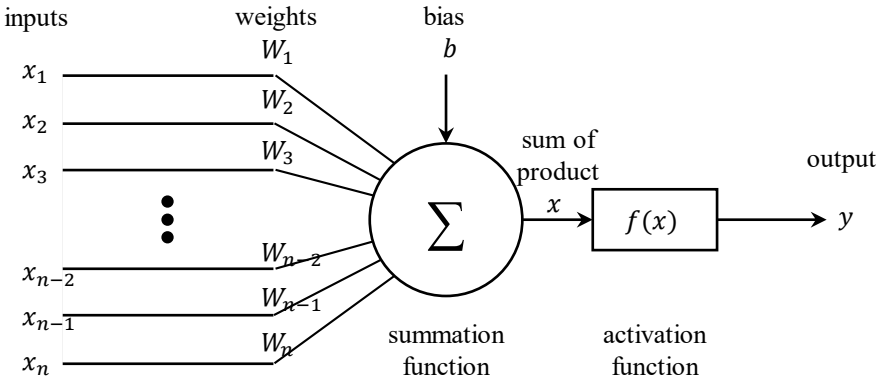
3.1.2. Structure of Neural Networks

There exist many variations of machine learning, one of them is based on the structure of artificial neural networks (ANNs) which were inspired by biological neural networks in animals, i.e. brains. An ANN consist of a network of simple elements which are called neurons. They are not only named after their biological counterpart but also are modeled according to the behavior of the neurons in our brain.

Figure 3.1 shows the structure of a biological and an artificial neuron. Just like the biological neuron, an artificial neuron can receive signals by a number of input channels. The equivalent in the biological structure are the dendrites. Both types of neurons have a processing stage,



(a) Structure of a biological neuron (source: [17]).



(b) Structure of an artificial neuron.

Figure 3.1.: The upper figure shows a biological neuron in simplified form. The lower one shows a single artificial neuron with n inputs (correspond to dendrites in biological neurons). The sum of the products of the weights and input values is handed to the activation function, which then results in one output value.

which is called cell body in Biology. The axon or simply the output in artificial neurons forwards the signals to other neurons.

Inside such a neuron a basic computation is conducted. The product of the input values x_n, i.e. the output of neurons of the previous layer, one constant bias $b = x_0$ (with $w_0 = 1$), and an equal amount of weights w_n are summed up

$$x = \sum_{i=0}^{n} w_i x_i \tag{3.1}$$

to form a weighted input value x. In the last step within a neuron, the so-called activation function takes this value as input and calculates the final output of the whole neuron. Section 3.2.1 gives more details about the activation function.

Figure 3.2 shows a small feed-forward network (signals flow in one direction, there are no loops in the signal path) with fully connected layers. The input layer is responsible for picking up the input signals and passing them on to the next layer, called hidden layer. There can be an arbitrary amount of hidden layers. In the end there is the output layer which delivers the result.

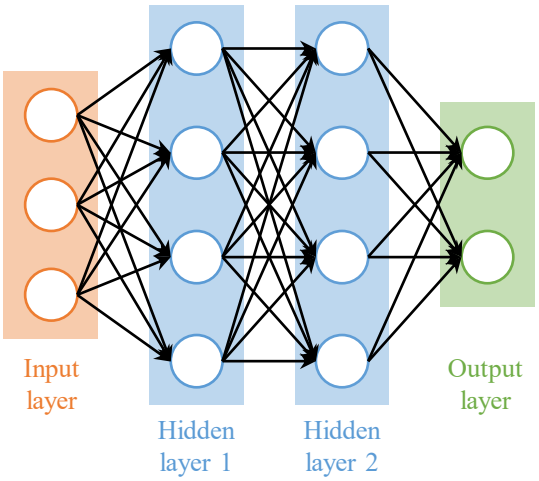


Figure 3.2.: A small artificial neural network with three input neurons, two output neurons and two hidden layers with four neurons each.

Just like human brains ANNs have to learn how to accomplish certain tasks. It cannot be programmed or configured to work in a predefined way as traditional algorithms. The next section describes three different learning techniques that neural networks use.

3.1.3. Learning Techniques

In machine learning there are three different learning techniques: supervised learning, unsupervised learning and reinforced learning. All of them are briefly described in this section.

Supervised learning

It is the most intuitive way of learning (for a human) and can be used if a large enough set of data with known results, i.e. labels, exists. We call this labeled or ground truth (GT) data. Typically a dataset is split into training, validation and test data. The algorithm analyzes the training data and produces a function with which new data can be mapped accordingly. In an optimal scenario the learning algorithm generalizes in a decent way, such that it correctly determines the class labels of unseen data. Usually the algorithm is applied to the test data, of which also the GT labels are available, in order to analyze the performance of the network.

Figure 3.3 shows a condensed overview on supervised learning. Features are extracted from the raw data on which the model is then trained, afterwards it is evaluated on the validation data. In contrast to the description above here the validation data instead of the test data is shown. The difference is, that with the validation data the algorithm is tested in the design phase. Meaning that a model is being trained, then evaluated on the validation set, and afterwards changes are being made to the algorithm according to the outcome of the performance, then the model is trained again. The test data is used at the end, when one model was chosen, in order to cross-validate and confirm the performance with another independent dataset (shown in the lower part of the figure).

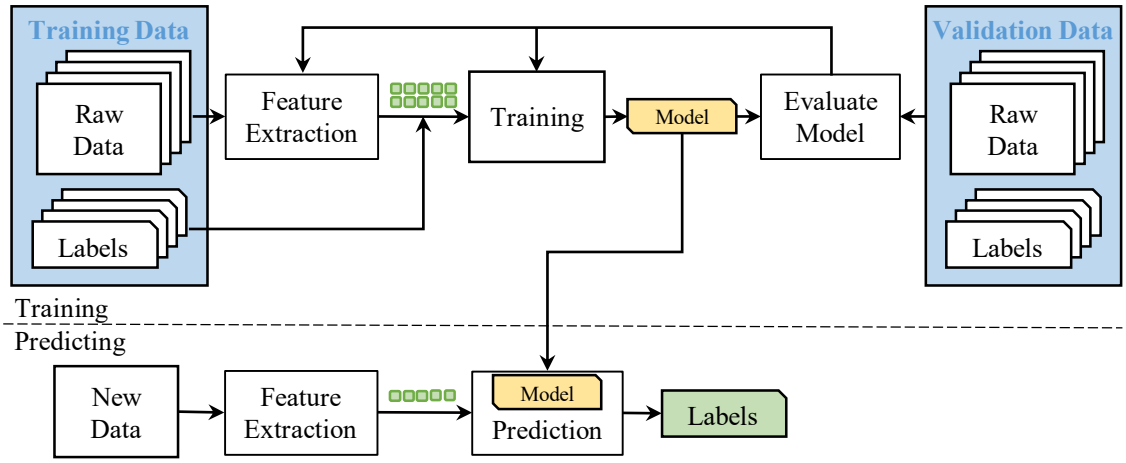


Figure 3.3.: The upper part shows a schematic of supervised training. In the lower part a trained model is applied to new data, i.e. the actual application case.

Supervised learning can be further divided into classification and regression. Whereas classification means that the output variable is a category such as "cat", "dog" or "disease", "no disease" and regression means that the output is a real value such as "meters" or "dollars".

Unsupervised learning

Unsupervised learning is useful if no labeled data is available, and if it is possible to derive some kind of cost function from the desired behavior. This tells the network how far apart it is from the target and can then adjust its parameters.

Figure 3.4 shows the difference between supervised and unsupervised learning. The three colors and shapes of the data points indicate labels, with which a supervised learning algorithm can calculate decision boundaries. In the right side of the picture all data points appear the same, as no labels exist, therefore an unsupervised algorithm groups them into clusters. Unsupervised learning can be further subdivided as well. One type is called clustering, where inherent groupings in the data is searched, e.g. grouping users by their online behavior. The other type is called association, here the algorithm should discover rules that describe large portions of the data, such as persons who like music A also tend to like music B.

Reinforced learning

Algorithms which allow software agents and machines to automatically determine their ideal behavior are using reinforcement learning. This happens within a specific context to maximize the performance. The algorithms are not given explicit goals, instead they focus to learn these optimal goals by trial and error.

Figure 3.5 shows that the reinforcement learning model is divided into two elements. The environment rewards the agent for correct actions (reinforcement signal) whereas the agent improves its environment knowledge to select the next action with the help of the obtained rewards.

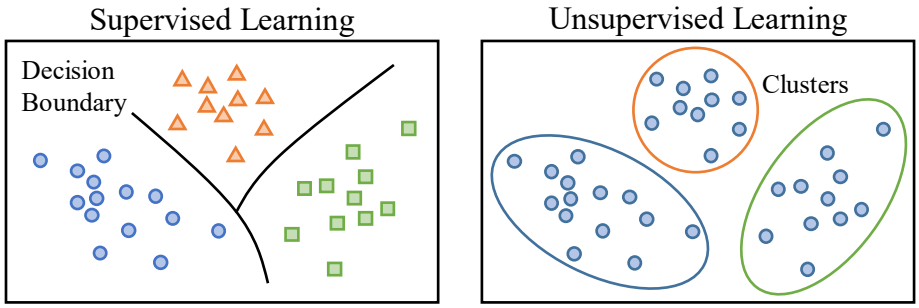


Figure 3.4.: Both sides show a problem with three classes. In supervised learning (left) the class labels are known (blue circle, red triangle, green square) and decision boundaries can be computed. In unsupervised learning however, no labels are know and the algorithm divides the data into three clusters.

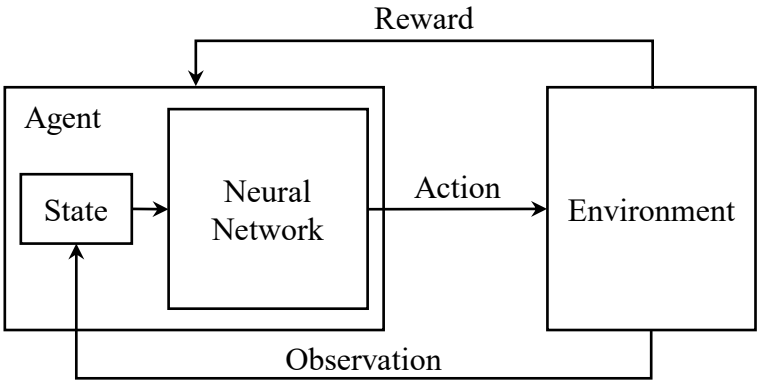


Figure 3.5.: Overview of reinforcement learning.

3.2. Network Components

This section explains all the important components of a NN structure. Section 3.2.1 discusses some useful activation functions, in section 3.2.2 backpropagation and optimizer types are introduced. Two kinds of network layers, fully-connected and convolutional, are described in section 3.2.3. The final section 3.2.4 explains pooling.

3.2.1. Activation Function

The activation function defines the output of a neuron given a weighted input value (refer to description of eq. 3.1). However, only nonlinear activation functions allow networks to compute nontrivial problems. There are several different activation functions available for various use-cases. In the following the ones used in this work and available in `Keras` are briefly presented.

Rectified Linear Unit (ReLU)

Figure 3.6 shows the transfer function of the ReLU [18] which is defined as:

$$f(x) = \max(0, x) \tag{3.2}$$

where x is the input of a neuron. It is the most popular activation function for deep neural networks (DNNs) [19]. Aside from being easy to calculate, ReLUs also have a reduced likelihood of the so-called vanishing gradient problem (see section 3.3.3), since the gradient stays constant. Krizhevsky et al. [20] showed that higher training speeds can be achieved in DNNs by using ReLUs. However, they can cause "dead" neurons, i.e. the output of a neuron is always zero, particularly if the learning rate (more in section 3.2.2) is set too high. Another restriction is that a ReLU activation function should only be used within the hidden layers of a NN.

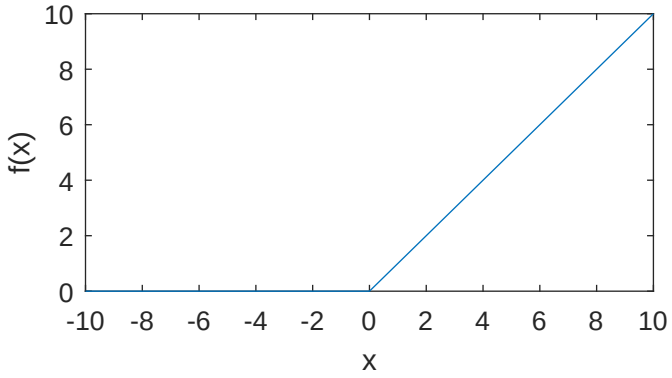


Figure 3.6.: ReLU transfer function

Softmax

The softmax activation takes a K-dimensional vector z as input and "squashes" it in a way such that all output values are in range [0,1] and sum up to 1. It is defined as

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad for \quad j = 1, ..., K \tag{3.3}$$

and its output can be seen as a probability distribution.

Figure 3.7 shows the transfer function of the softmax (logistic) function

$$h_\theta(x) = \frac{1}{1 + exp(-\theta^T x)} \tag{3.4}$$

where θ represents a vector of weights and x is a vector of input values. This function is used to approximate the target function $y \in \{0, 1\}$ in binary classification. The softmax function produces a scalar output $h_\theta(x) \in \mathbb{R}$, $0 < h_\theta(x) < 1$.

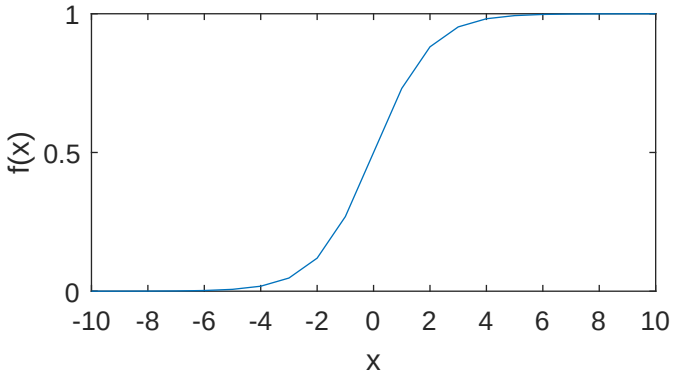


Figure 3.7.: Softmax transfer function

Further activations

The sigmoid function is a special case of the logistic function and is defined as

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

It is a S-shaped curve and ranges between 0 and 1. Even though it is easy to understand and apply, it got less popular. Some of the reasons are, that it suffers from the vanishing gradient problem and has a slow convergence.

Another function is the Hyperbolic Tangent function (tanh), its formula is written as

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}. \tag{3.6}$$

As it ranges in between -1 and 1, i.e. -1 < output < 1, the output is zero centered. Hence optimization is easier than for sigmoid, thus this method usually is preferred over the sigmoid function in practice, even though it still suffers from vanishing gradient problem.

A smooth approximation to ReLU is the analytic function

$$f(x) = log(1 + e^{x}), \tag{3.7}$$

called softplus function [21]. The derivative of softplus is the logistic function $f'(x) = \frac{1}{1+e^{-x}}$.

3.2.2. Training Process

In classification with supervised learning, the goal is to generate as many correct labels as possible with the network, while still being able to generalize. This means that a similar result can be obtained for data which was not used for training before.

Backpropagation

The goal of backpropagation [22] is to optimize the weights in a manner, that the NN can learn how to correctly map arbitrary inputs to outputs. In a feed-forward network there are

no cyclic connections between neurons, instead each neuron can only be connected to the neurons of the next layer. During training, the input is first propagated forward through a network, called the forward pass. At the last layer, first the loss of each output neuron is calculated with a certain loss function and afterwards it is propagated backwards through the network, until each neuron has a corresponding loss value. This is called the backwards pass.

The Forward Pass At first we feed inputs forward through the network which has initial weights and bias values. Afterwards the total net input to each hidden layer neuron is computed and "squashed" using an activation function. This process is repeated with the output layer neurons until the last layer is reached. Here the total error is computed by calculating the error for each output neuron using the squared error function

$$Error_{total} = \sum \frac{1}{2}(target - output)^2 \tag{3.8}$$

where target is sometimes also called *ideal* and output is referred to as *actual*.

The Backwards Pass The goal here is to update each of the weights in the network in order to approach the target output by the actual output by minimizing the error for each output neuron and the network as a whole.

Figure 3.8 shows a neuron in the output layer. Considering w_3 we want to know how much a change in w_3 affects the total error, $\frac{\partial E_{total}}{\partial w_3}$. What is shown visually can be described by the chain rule with

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{o1}} \cdot \frac{\partial out_{o1}}{\partial net_{o1}} \cdot \frac{\partial net_{o1}}{\partial w_3} \quad . \tag{3.9}$$

To decrease the error, this value is then subtracted from the current weight, optionally multiplied with some learning rate η

$$w_3^+ = w_3 - \eta \cdot \frac{\partial E_{total}}{\partial w_3} \quad . \tag{3.10}$$

The actual updates in the network are performed after the new weights leading to the hidden layer neurons are obtained, i.e. use the original weights, not the updated weights and then continue the backpropagation algorithm.

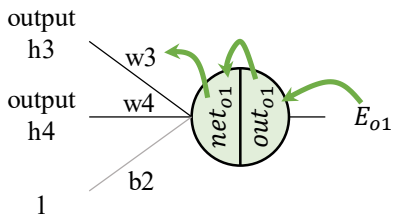


Figure 3.8.: Output neuron in the backward pass

Figure 3.9 shows a neuron in the hidden layer which calculates the new weights similar to the output neuron $\frac{\partial E_{total}}{\partial w_3}$. What is shown visually can be described by the chain rule with

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} \cdot \frac{\partial out_{h1}}{\partial net_{h1}} \cdot \frac{\partial net_{h1}}{\partial w_1} \tag{3.11}$$

with

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} \tag{3.12}$$

and $E_{total} = E_{o1} + E_{o2}$.

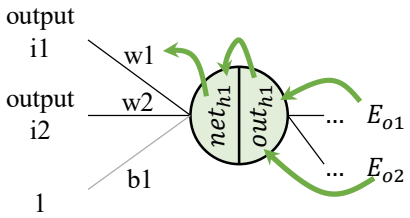


Figure 3.9.: Hidden neuron in the backward pass

The output of each hidden layer neuron contributes to the output (and therefore error) of multiple output neurons, that is why the calculation for the hidden neurons is slightly different. out_{h1} affects both out_{o1} and out_{o2}, therefore the $\frac{\partial E_{total}}{\partial out_{h1}}$ needs to take into account its effect on both output neurons. w_1 is then updated the same way as w_3 shown above and finally all weights are updated. After every round of backpropagation the error decreases, causing the output to approximate the target.

In general, the losses of the neurons of the hidden layers are basically indicators on how influential the individual neurons were on the end result. These can then be used to calculate the gradients, which are utilized later on to optimize the parameters of the network.

Optimizer

In general an optimization algorithm minimizes or maximizes an objective function (error function) which is simply a mathematical function dependent on a model's internal learn-able parameters. In NNs the goal of an optimizer is to adjust the parameters of the network in order to minimize the loss function. The most common approach is the gradient descent technique. As described above, after retrieving the gradients of a a model through backpropagation, the weights are updated in the opposite direction of the gradient of the loss function to move closer to the local minima. An additional parameter called learning rate is added to better regulate the amount of adjustment during the training process. Generally its value is reduced over time to reduce the risk of obtaining a sub-optimal local minimum instead of a the desired global one. The higher the parameter update, the higher the "jumps" in the loss function. This is useful in the beginning of a training to step over local minima, but later this might lead to a "ping-pong"-like effect with no convergence if the update rate is not reduced over time.

Fig. 3.10 shows such a "ping-pong"-like effect.

`Keras` provides several different optimizers, a short list is shown below:

- Stochastic Gradient Descent (SGD)
- Adaptive Gradient Algorithm (AdaGrad)
- Root Mean Square Propagation (RMSprop)
- Adaptive Moment Estimation (Adam)

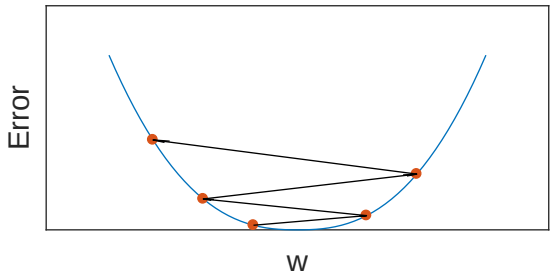


Figure 3.10.: Example: the resulting loss oscillates around the minimum due to a too high learning rate.

SGD [23] is a variant of the standard gradient descent in which a parameter update occurs with only one randomly picked training sample instead of the whole dataset. This results in more fluctuations within the loss function, which provides better local minima and the possibility to train using large-scale datasets. AdaGrad is a modified stochastic gradient descent with per-parameter learning rate [24]. RMSprop has shown excellent adaptation of learning rate in different applications and is capable to work with mini-batches as well opposed to only full-batches. Adam is an update to the RMSprop optimizer which uses running averages of both the gradients and the second moments of the gradients [25].

3.2.3. Network Layers

This section introduces two types of layers, i.e. fully-connected and convolutional, used to form a NN. Since both of them perform the same forward pass operations internally, they can easily be converted from one to the other. For instance, a conversion from convolutional to fully-connected would result in a weight volume which is largely zero because of the local activation areas of the convolutional layer, while many weights would also be equal because of the weight sharing. This will become clearer in the following sections.

Fully-Connected Layers

The basic neural network layer is the fully-connected layer. It simply connects all neurons of the input with those of the output, as shown in fig. 3.2. However the amount of parameters easily rises to unmanageable sizes and DNNs consisting purely of fully-connected layers quickly reach the limits of current hardware in terms of memory.

Convolutional Layers

The structure of a classic CNN principally consists out of one or more convolutional layers followed by a pooling layer (section 3.2.4). This structure can be repeated any number of times. When repeated sufficient enough it is called a deep convolutional neural network.

Convolutional layers are the main building block for NNs performing image classification nowadays. They mimic the way a cat's visual system works by using overlapping areas of sensor signals as input values [26, 27]. In the first layers, it detects local visual features like

edges or corners, which are then combined by subsequent layers. The ability to take into account local distortions of the input makes them invaluable for spatial input like images. Furthermore, they only require a fraction of the weights used by a standard fully-connected layer by using a method called weight sharing. The reason is, it is assumed that if a local feature detector was computed on one part of the image, it is most likely useful on the entire image.

Figure 3.11 shows that the neurons of a convolutional layer are arranged in three dimensions (width, height and depth) which are similar to the layout of an image with dimensions x, y and color channels. In each layer, a 3D input volume of neurons is transformed through a differentiable function to form a new 3D volume. For instance, the input volume at the first layer could be an image of size 32x32x3, and the output of the last layer a volume of size 1x1x12, assuming there are twelve classes available for classification.

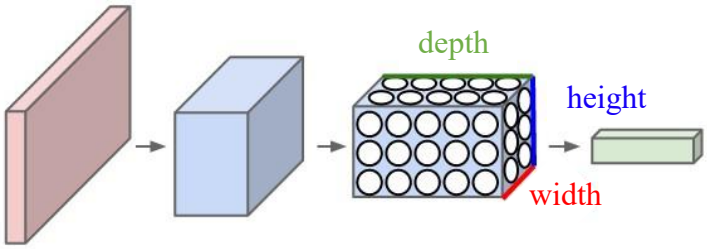


Figure 3.11.: A CNN arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers (source: [28]).

Each convolutional layer also has a set of trainable filters which are arranged in three dimensions and contain the individual weights. However their width and height are relatively small, while the depth is equal to that of the corresponding depth of the input volume, i.e. the number of feature maps. Common filter sizes range from 1x1xd to 7x7xd, d being the depth.

Figure 3.12 shows that these filters are moved with a certain step size across the input volume (in the forward pass). The dot product of the filter values and the input results in a 2D activation map. Those activation maps of each filter are combined and form the output volume. Additionally, so-called zero-padding can be applied to control the output dimension.

Assuming we have an input volume of size w_i x h_i x d_i given n filters of size f_x x f_y with step size s and zero-padding of p, the output volume has the size

$$w_o = \left\lfloor \frac{w_i - f_x + 2p}{s} \right\rfloor + 1 \quad , \quad h_o = \left\lfloor \frac{h_i - f_y + 2p}{s} \right\rfloor + 1 \quad , \quad d_o = n \quad . \tag{3.13}$$

It is important to have a valid set of hyper-parameters. For instance, a 10x10x3 input volume cannot be traversed by a 3x3 filter with step size 2, as the filter would step out of bounds.

3.2.4. Pooling

Another essential part of a CNN is the pooling layer. It is usually inserted between convolutional layers in order to reduce the spatial size and therefore the amount of parameters in the network. This serves the purpose of reducing computational costs and to avoid possible

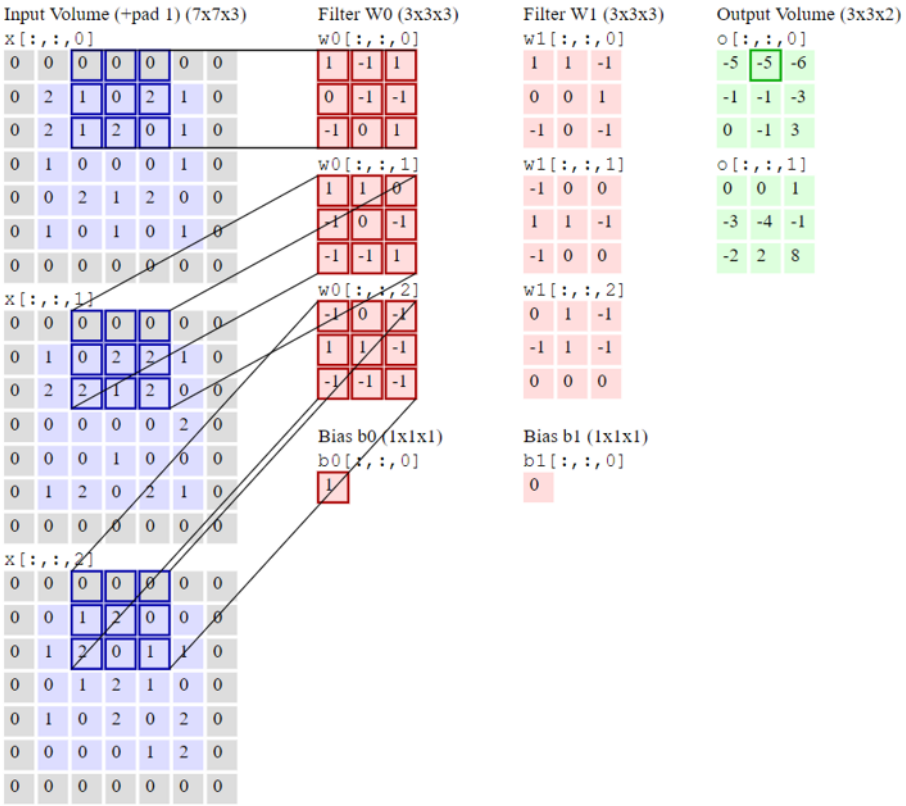


Figure 3.12.: Two filters with step size 2 are applied to an input volume of size 7x7x3 with zero-padding 1 (source: [28]).

overfitting. Pooling layers have a size p_x x p_y and a certain step size s as hyper-parameters. They resize every input depth slice by simply traversing it with s and calculating either the average or the maximum of a local area with size p_x x p_y.

Figure 3.13 shows an example of maximum pooling with size 2x2 and step size 2 on a single depth slice.

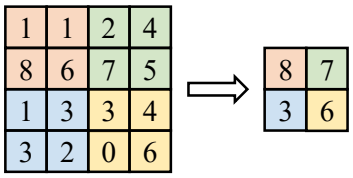


Figure 3.13.: Pooling example (source: [28])

3.3. Common Network Effects

3.3.1. Batch Normalization

Ioffe and Szegedy introduced Batch Normalization (BN) in 2015 [29] in order to reduce the effect of varying input distributions within training batches on the network parameters. Nor-

malization (shifting inputs to zero-mean and unit variance) is often used as a pre-processing step to make the data comparable across features. As the data flows through a deep network, each layer constantly has to adapt to new input distributions, sometimes making the data too big or too small again. This problem is referred to as "internal covariance shift" [30]. These shifts amplify the deeper the network.

By normalizing each input of the activation function by both mean and variance this effect can be reduced. This means, instead of performing normalization once in the beginning, it is done all over the network for each mini-batch. However, the way to normalize the data differs for training and testing. In training mode the batch normalization of sample x_i is calculated as

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{3.14}$$

with batch mean $\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$ and batch variance $\sigma_B = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$. ϵ is added to avoid dividing by zero. Since the input of the activation function is limited to a normal distribution with the current setup, the representational power of the layer would be reduced. Therefore the two trainable parameters scale γ and shift β are added, which reverse some of the batch normalization

$$y_i = \gamma\hat{x}_i + \beta = \gamma\frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta \quad . \tag{3.15}$$

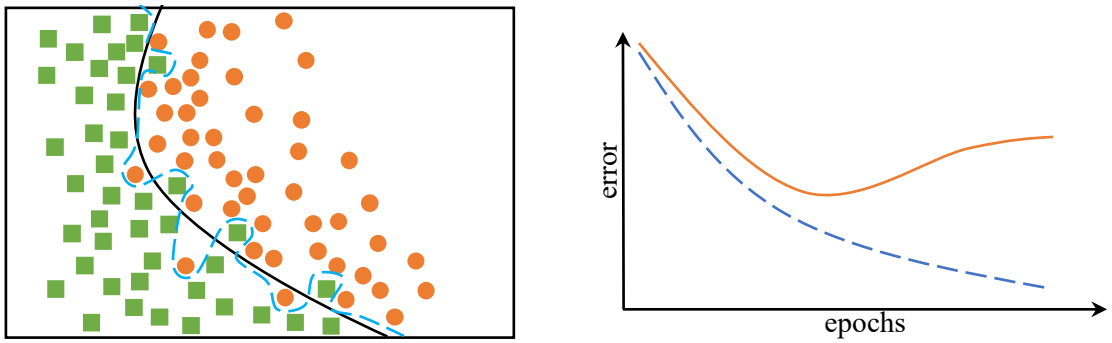
In order to predict individual samples, population statistics need to be used in testing/predicting instead of batch statistics. This is because otherwise the input of the activation function would always be zero, i.e. the mean of the "batch" of a single sample. To avoid this, the estimated population statistics need to be calculated using an exponential moving average of the batch statistics during training and saved for later testing.

DNNs converge faster and better when batch normalization is added. Furthermore the correct initialization of the weights and biases of the network, which is a quite difficult and complicated task, becomes less influential on the end result.

3.3.2. Overfitting

Overfitting is a common problem in training NNs. It means that the network is too specialized on certain data, such that it cannot generalize well and thus performs bad on unseen data.

Figure 3.14 shows such a problem. In Fig. 3.14a the decision boundary (dashed, blue line) of a two class separation problem is too complex. It follows the training data and thus is too dependent on the data. It is more likely to have a higher error rate on unseen data than with the black decision boundary. Fig. 3.14b shows how this effect can be spotted. Epochs are the "steps" or "cycles" of the incremental parameter updates explained in section 3.2.2. Usually both training and testing error should decrease over a time of training. In case the training error is low, but the testing error is still high, this means that the network specialized too much on the training data. There are several methods to counteract this phenomenon, they are explained in the following sections.



(a) Decision boundaries for an overfitted (dashed blue line) and an regularized model (black line)

(b) Error rate for training (dashed blue) and test data (orange)

Figure 3.14.: Example for overfitting

L2 Regularization

In general, regularization is a process of introducing additional information in order to avoid overfitting. L2 Regularization is the most common type of weight decay. For every weight w the term $\frac{1}{2}\lambda w^2$ is added to the loss function, with λ being the decay intensity. Therefore the gradient with respect to w is simply λw (from $\frac{\partial}{\partial w}\frac{1}{2}\lambda w^2 = \lambda w$). This value is then subtracted from the weight during the parameter update, decaying linearly towards zero, which helps with reducing overfitting in general.

Dropout

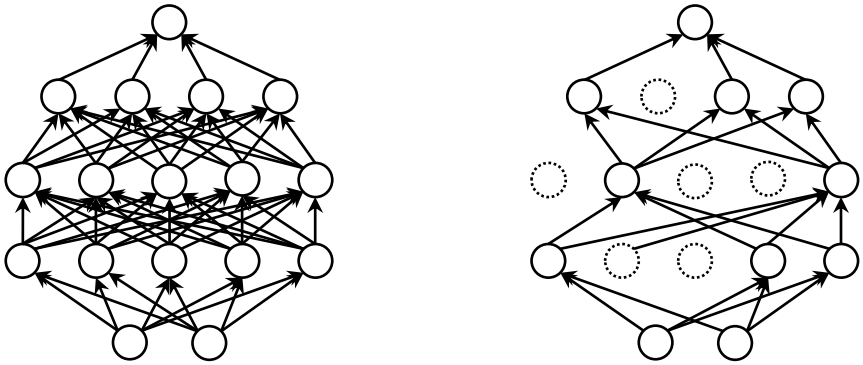
Dropout [31, 32] is another technique to reduce the problem of overfitting. At each training stage individual nodes are "dropped out" of the net, i.e. temporarily removed with the probability $1 - p$ or kept with probability p, such that a reduced network emerges. Incoming and outgoing edges to a dropped-out node are also removed. This prevents so-called co-adaption, i.e. neurons becoming too dependent on other specific neurons.

Figure 3.15 shows the same NN before and after applying dropout. Only the reduced network is then trained on the data at that stage, afterwards the removed nodes are reinserted into the network with their original weights.

3.3.3. Vanishing Gradient Problem

The vanishing gradient problem [33] occurs in ANNs with gradient-based learning methods and backpropagation. Since increasingly smaller gradients are multiplied by the chain rule (as explained in section 3.2.2), the front layers of the network may receive an infinitesimal or even no update at all. In the worst case, this may stop the network from further training. The deeper it is, the more it suffers from this difficulty. Further the activation function plays a major role, since for instance the sigmoid activation

$$S(x) = \frac{1}{1 + e^{-x}} \tag{3.16}$$



(a) Standard neural network (b) Network after applying dropout

Figure 3.15.: Applying dropout to a neural network

results in smaller gradients, the higher the value of its input. The ReLU activation function, as opposed to this, circumvents this problem since the gradient is not bound to move towards zero.

3.3.4. Class Imbalance

In many applications it can happen that a dataset contains a lot of samples from one class, but only a few of another class. For a binary classification problem, such as healthy and diseased patients, the diseased patient is the "interesting" class, but might only make up 10 % of the dataset. This can cause the network to not properly train and simply assume in classification that all samples belong to the class of healthy patients. Even though all classifications for diseased patients are incorrect, the network would still obtain an accuracy of 90 % due to

$$\text{accuracy} = \frac{\text{number of correctly predicted samples}}{\text{total number of samples}} = \frac{0.9 \cdot N}{1 \cdot N} \cdot 100\% = 90\% \tag{3.17}$$

with N being the number of samples. This means, that either a dataset with balanced classes should be used, or that the training needs an additional information about the class weights. In `Keras` this can easily provided via a dictionary.

4. Proposed Methods

This chapter describes the development environment and its software and hardware requirements used to conduct this work. In section 4.3, the network architecture and why it was chosen is explained. Furthermore, it describes the evaluation metric to test the architecture and validate the results to make them comparable.

For this work the open source high-level neural networks application programming interface (API) `Keras` [34] was used. It is written in Python and running on top of `TensorFlow`. The following will provide more detailed information about this.

4.1. Development Environment

4.1.1. Hardware requirements

It is a common misconception that large data centers are needed to run deep learning algorithms, they can even be conducted on laptops. However, in general it can be said, the smaller the system, the more time is needed to train a model with a good performance. Deep learning requires a lot of computational power to run, among all the different tasks inside, training of the model is the most intensive one.

As explained above, the forward and backward pass consist of many matrix multiplications. For e.g. VGG16 [35], a CNN with 16 hidden layers and about 140 million parameters (weights and biases), it would take years to train considering all the matrix multiplications in a traditional approach. By doing all computations at the same time, the training time can be reduced tremendously, which is why parallelization with graphics processing units (GPUs) is important for machine learning. Since we use `TensorFlow` in this work, `CUDA`® is required to be installed on our system which needs a `NVIDIA`® graphics card to run.

Table 4.1 shows the hardware properties of the system used for this work, more details on the particular components are given in appendix A.3. This work was conducted on a single local Linux machine running Ubuntu 17.04.

Table 4.1.: Hardware properties of the system

	details
memory	15.5 GiB
processor	Intel® Core™ i7-7700CPU @ 3.60GHz x 8
graphics	GeForce GTX 1070

4.1.2. Software requirements

As mentioned above, CUDA® is installed on the system in order to run TensorFlow with GPU support, which makes the program significantly faster. Furthermore, the CUDA® Deep Neural Network library (cuDNN) is used, which is a GPU-accelerating library [36].

Table 4.2 shows the versions of the software components and the operating system (OS) of the machine.

Table 4.2.: Software versions of the system

	version
Linux Ubuntu	17.04 (64-bit)
CUDA	8.0.61
cuDNN	6.0
TensorFlow	1.3.0
Keras	2.0.8

4.1.3. Libraries

The key aspect of this work is, that the deep learning problem should be solved with Keras which runs on top of TensorFlow in our case. Both frameworks are shortly described below.

TensorFlow TensorFlow is an open-source software library developed by Google and was released by the end of 2015. It is available for Linux (64-bit), macOS and Windows, as well as for Android and iOS. It can run on multiple CPUs and GPUs with optional CUDA® extensions, as we use it in this work. The name TensorFlow derived from the multidimensional data arrays that are processed in NNs and are referred as *tensors*. For more detailed information about TensorFlow and the Google network, refer to [37].

Keras Like TensorFlow, Keras is an open-source neural network library written in Python. It can run on top of TensorFlow, as in our case, or with other libraries, e.g. Theano. Developed by the Google engineer François Chollet it enables fast experimentation with DNNs. For more information refer to the Github repository [34].

4.2. Prostate Cancer Dataset

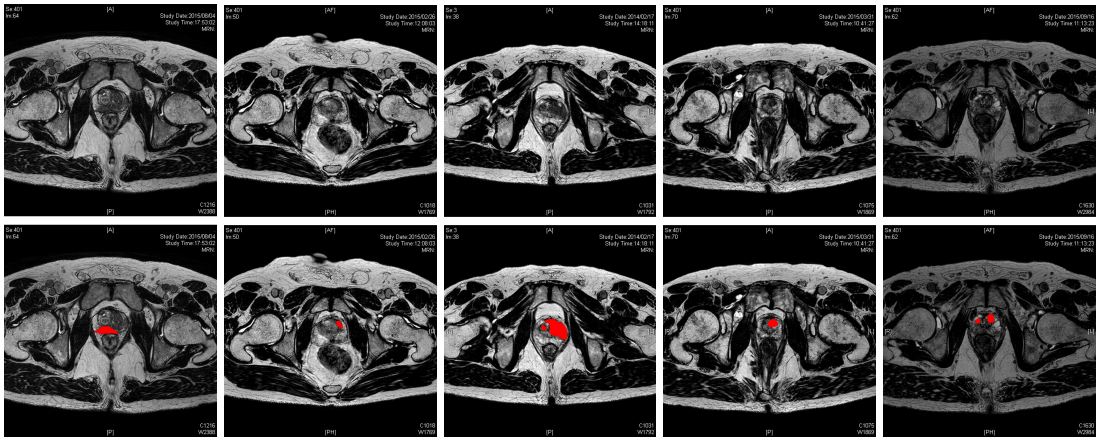
This work is based on the prostate cancer dataset provided through the partnership of *Tokyo Institute of Technology* and *Tokyo Medical and Dental University*. In this section a very basic introduction to the dataset is given, showing some example images. The technical and medical descriptions of the dataset can be found in the paper by Numao et. al [38].

The dataset contains 1465 images of 218 patients that suffer from prostate cancer and 3460 images of 128 healthy patients. The amount of scans per patient differs. For each scan with

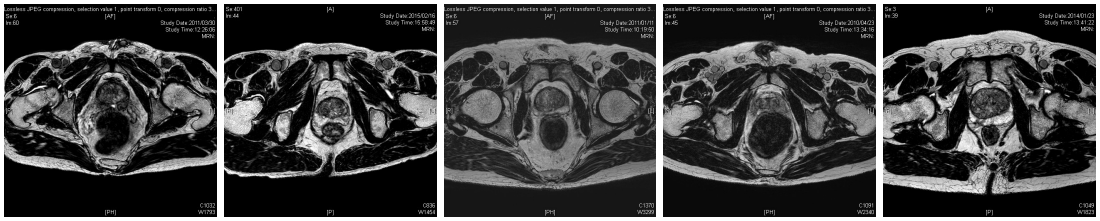
cancerous tissue, an annotated image exists. Most of the images have the size 512x512 pixels and obtain 3 channels with 8 bits. However, some images are smaller or have an additional alpha channel, which means data preprocessing for equalization and normalization is needed before training.

Figure 4.1 shows some example images of the dataset. Fig. 4.1a displays scans of cancer patients, where the upper row are the original scans and the lower row the scans with the annotated cancer region. Fig. 4.1b shows examples of five healthy patients. In all scans, the prostate is more or less located at the center of the image and the bodies are scaled to an approximately consistent size. The brightness, however, is not utterly constant in all images, evidently seen in the rightmost images of cancer cases. Also the contrast may differ in some of the scans, as visible e.g. in the middle image of the healthy cases.

Unfortunately, the dataset does not provide ground truth information about segmentation of the prostate, therefore no prostate segmentation is done in this work for preprocessing, unlike in most other publications as described in chapter 2. In the future, *Tokyo Medical and Dental University* will provide more MR images and labeled data, and also include the annotations for prostate segmentation.



(a) Cancer cases (top row) with their annotations (bottom row)



(b) Healthy cases

Figure 4.1.: Example images of cancer cases with their annotations and healthy cases of the prostate dataset provided by *Tokyo Medical and Dental University* [38]

For the training and evaluation process the dataset was split into training, validation and testing. The exact numbers are listed in table 4.3. As the number of samples per class, i.e. number of images for cancer and healthy cases, is not equal, the network can suffer from class imbalance problems, as described in section 3.3.4. Hence, either a "class-balanced" dataset or a class weight dictionary is used for training in this work.

Table 4.3.: Split of the dataset into training, validation and testing

	fraction	number of images	
train	0.65	955	cancer
		2261	healthy
val	0.15	220	cancer
		521	healthy
test	0.20	290	cancer
		678	healthy

In case of the "class-balanced" dataset, the split of the prostate dataset is 955 training, 220 validation and 290 test samples for healthy and cancer patients, respectively. In case of defining class weights, the dictionary is passed to the training function as follows:

```
class_weight = {
  0: 1.,
  1: num_cancer_cases / num_healthy_cases
}
```

It is used for weighting the loss function in training only, in order to "pay more attention" to the samples of cancer cases. For validation this does not apply.

4.3. Implementation

4.3.1. Architecture

The initial idea of this work is to perform classification of MR scans into prostate cancer and healthy patients with a simple neural network. This decision has several reasons.

First, the general risk of overfitting increases with network size, when using a relatively small dataset like in this work. The reason is, that it is quite easy to build a complex network that perfectly fits the data, but on the other hand does not generalize well, i.e. performs poorly on new data. Furthermore, the training time increases the bigger and more complex the network gets. To start developing with a smaller network can show faster results, which makes it easier to make further adaptations and expand the network gradually, such that it performs in a certain way. And finally, understanding a large and complex network can be very costly with the risk of not fully comprehending what the network is actually learning. This can cause mistakes, in a way that the network is extracting information from the data that is not related to the actual problem, but might still cause the network to perform in such a way that it appears to do the job correctly.

Figure 4.2 shows the initial network structure. It is composed of a stack of three convolutional layers (dashed black boxes) with a ReLU activation and max-pooling layer, followed by two fully-connected layers. As input it expects an image of the size 300 x 300 pixels with three channels. All images are rescaled with a factor of $\frac{1}{255}$ before usage to obtain pixel values in the range [0, 1). The first part of the model with the three convolutional layers outputs 3D feature maps (height, width, features) which are then converted to a 1D feature vector by the flatten layer. The final output of the network is a probability vector for each image. It gives the probability per class, in our case two classes: cancer cases and healthy cases.

A dropout layer is added to the model. As described in section 3.3.2, dropout helps to reduce overfitting by preventing a layer from seeing the exact same pattern twice. It tends to disrupt random correlations occurring in the data.

After defining the architecture, the model is compiled with a `categorical_crossentropy` loss function (objective function/optimization score function) and RMSprop optimizer with a learning rate of 0.001. Alternatively `binary_crossentropy` can be used, as this is a binary classification problem.

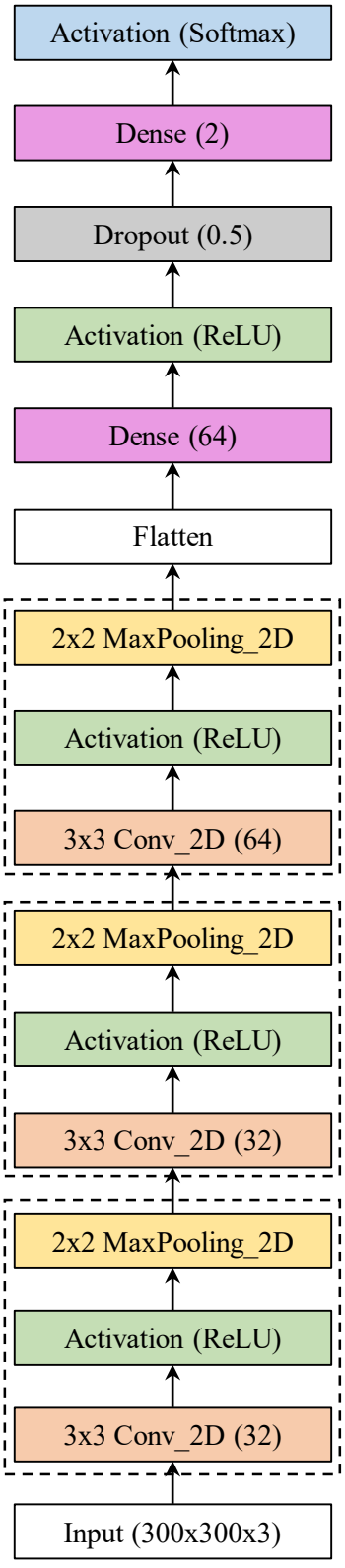


Figure 4.2.: Initial network architecture

The described model can be created with `Keras` as follows:

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras import losses, optimizers, metrics

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(300, 300, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('softmax'))

model.compile(loss=losses.categorical_crossentropy,
              optimizer=optimizers.rmsprop(lr=0.001),
              metrics=[metrics.categorical_accuracy])
```

`Keras` provides generators for on-the-fly data preprocessing and augmentation. This can be helpful in order to extend the dataset without the model seeing the exact same image several times, which is again useful to prevent overfitting. It can be done with the `keras.preprocessing.image.ImageDataGenerator` class, with which random transformations and normalization operations can be performed on the image data during training. Furthermore, it can instantiate generators of augmented image batches and their labels, which can be used as input for the `Keras` model. The following code shows an example for data augmentation, as it is used in the experiments of section 5.6.

```
from keras.preprocessing.image import ImageDataGenerator

# augmentation configuration for training
train_datagen = ImageDataGenerator(
    rescale=1. / 255,  # applied to all trainings in this work
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1
)

# generator that will read pictures found in subfolers
# of 'train_data_dir', and indefinitely generate
# batches of augmented image data and their labels
```

```
train_generator = train_datagen.flow_from_directory(
    train_data_dir,  # target directory
    target_size=(200,200),  # all images will be rescaled to 200x200
    classes=['cancer_cases', 'healthy_cases'],
    batch_size=128
)
```

The rescaling of the pixel values in the images, to a range of [0, 1), is done with the `ImageDataGenerator` for all images, as indicated in the code above. With the `.flow_from_directory()` function, the image data batches with the labels can directly be generated from the files on the harddrive. For this a training data directory and a validation data directory containing one subdirectory per image class, filled with .png or .jpg images, has to exist. In our case there is a third directory for test data, which will be used at the very end (not shown below).

```
data/
    train/
        cancer_cases/
            0001.jpg
            0002.jpg
            ...
        healthy_cases/
            0001.jpg
            0002.jpg
            ...
    val/
        cancer_cases/
            0001.jpg
            0002.jpg
            ...
        healthy_cases/
            0001.jpg
            0002.jpg
            ...
```

The batch size can be chosen arbitrarily, the default value in `Keras` is 32. In case of a dataset with 1000 training samples and a batch size of 100, the algorithm takes the first 100 samples and trains the network. Next it takes the second 100 samples and trains the network again. This procedure is repeated until all samples are propagated through the network. The advantage is that the overall training procedure requires less memory since the network is trained using a smaller number of samples per run. This is particularly important if the dataset does not fit into the memory. Typically, networks train faster with mini-batches, as weights are updated after each propagation. In this example, 10 batches are propagated through the network (10 batches with 100 samples = 1000 samples) which conforms to 10 network parameter updates. If all samples were propagated through the network at once, there would be only one update per network parameter. The downside is, the smaller the batch, the less accurate the estimate of the gradient is. In this work a batch size of 128 is being used for all experiments, except the ones in section 5.7.

4.3.2. Evaluation metric

The prostate dataset is split into three sets: training, validation and testing, according to the values in section 4.2. During the training phase, the training data is presented to the model, which is trained by pairing the input with the expected output. The validation phase is used to estimate how well the model has been trained and how it performs (e.g. mean error for numeric predictors, classification errors for classifiers, recall and precision for IT-models, ...). It is usually split into two parts.

In the first part, the model that performs best on the validation data, is selected. Then, the accuracy of the selected approach is estimated with the test data. In case it is not necessary to choose an appropriate model from several approaches, it is sufficient to only have a training and test set, without performing validation of the trained model. A typical machine learning task can be visualized as in algorithm 1:

Algorithm 1 Nested loop for machine learning task

```
while error in validation set > x do
    tune hyper-parameters
    while error in training set > y do
        tune parameters
    end while
end while
```

Typically, the outer loop is performed by a human on the validation set, and the inner loop by a machine on the training set. In other words, the validation set is the training set for humans. Afterwards the test set is needed to assess the final performance of the model.

The `.fit()` method of `Keras`, i.e. the training, can additionally take validation data as an input and so automatically output the validation accuracy after each epoch in addition to the training accuracy. However, in order to be sure the validation is correct, it is useful to make an additional validation of the model afterwards. This can be done via `.evaluate_generator()` or by predicting the labels of the data with `.predict_generator()` and then calculating the accuracy by hand with the output results and the given labels. As `Keras` is a high-level API, it is sometimes useful to go slightly deeper in order to better understand what is going on and if the program does what it is designed for. `Keras` offers callbacks to save information or affect the training process for this purpose. The callbacks are briefly described in the next section.

In the results chapter 5 not only the accuracy and loss are presented, which are automatically given by `Keras` but also precision, recall and f1-score. All three of them are computed with the help of scikit-learn, which is a simple and efficient tool for data mining and data analysis in Python [39]. With the `classification_report` precision, recall and f1-score can easily be obtained, only the classes and the predictions are necessary as inputs.

```
import sklearn.metrics as skm
import numpy as np

# obtain prediction vector for the samples
pred_probablity = model.predict_generator(test_generator)
# convert the prediction probabilities into class predictions
```

```
pred_classes = np.argmax(prediction_probability, axis=1)
print(skm.classification_report(test_generator.classes, pred_classes))

>>>                  precision  recall   f1-score   support
>>>
>>> cancer_cases        0.99     0.84      0.91       290
>>> healthy_cases       0.94     1.00      0.96       678
>>>
>>> avg / total         0.95     0.95      0.95       968
```

For reference, when the model is applied to a single image, it outputs a vector that looks like this: [0.7435, 0.2565]. It means, that the network assigns this input to the class cancer cases with a probability of 74.35% and to healthy cases with a probability of 25.65%. The decision threshold for the mentioned evaluations lays at 50%.

In general, precision and recall are defined as

$$precision = \frac{TP}{TP + FP} \tag{4.1}$$

$$recall = \frac{TP}{TP + FN} \tag{4.2}$$

with values obtained from a confusion matrix as in table 4.4. Positive instances, in this case, are cancer cases, while healthy cases are defined as negative instances.

Table 4.4.: Structure of a confusion matrix

		ground truth	
		cancer	healthy
predicted	cancer	true positive (TP)	false positive (FP)
	healthy	false negative (FN)	true negative (TN)

In this work the model is optimized for accuracy, which can as well be calculated by the results of the confusion matrix

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \tag{4.3}$$

The balanced F-score is a harmonic mean of precision and recall

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall}. \tag{4.4}$$

It is approximately the average of precision and recall when they are close. Here, it is also know as the F1-score, because recall and precision are evenly weighted.

4.3.3. Keras callbacks

A callback is a set of functions to be applied at given stages of the training procedure. It can be used to get a view on internal states and statistics of the model during training. Several

callbacks can be generated and passed to the `.fit()` method as a list, where the relevant methods will then be called at each training stage.

It is useful to include `ModelCheckpoint` into the callbacks, as it can save the weights (or even the whole model) either after each epoch or only if the accuracy has improved. So it is easy to evaluate a model at a specific epoch after the training has finished. The `CSVLogger` streams epoch results to a csv file which is helpful in order to generate accuracy and loss curves. `TensorBoard` does something similar, it writes a log file which allows visualization of dynamic graphs of training and validation metrics and activation histograms for different layers in the model. All these callbacks are used in this work to supervise the training, and evaluate the results.

Further useful callbacks are `EarlyStopping` and `LearningRateScheduler`. `EarlyStopping` stops the training when a monitored quantity, i.e. validation loss, stopped improving. A patience interval can be set for this, e.g. if set to four, it waits four epochs before stopping the training in case no improvement occurred. This is useful in the "trial-and-error phase" at the beginning of a project. The `LearningRateScheduler` is a useful tool to decrease the learning rate over epochs, however, it is important to not cause overfitting with this, as a very small learning rate can lead the algorithm to fall into a local minimum. The callback expects a function as input that takes an epoch index as input and returns a new learning rate (example below).

```
from Keras.callbacks import LearningRateScheduler
import math

def step_decay(epoch):
    initial_lrate = 0.01
    drop = 0.5
    epochs_drop = 6
    lrate = initial_lrate * math.pow(drop, math.floor((1 + epoch) /
                                    epochs_drop))
    return lrate

callbacks_list = [LearningRateScheduler(step_decay)]
```

5. Experiment Results

Figure 5.1 shows an overview of this chapter, which consists out of eight sections. Section 5.1 describes the experiment setup and the starting point for the achieved performance. The following six sections present training results of different experiments and the last section, 5.8, summarizes and discusses the preliminary findings.

Section 5.2 examines weather the imbalance of the number of training samples per class has an influence on the performance. In section 5.3 the classification results for several input image sizes and extend of the cropped region are compared. Three different optimizer types with a variety of learning rates are tested in section 5.4 and section 5.5 investigates the influence of the activation function in the output layer, and additionally compares the training results with varying dropout values. Section 5.6 shows weather data augmentation achieves any advantages or not in this type of application. The last of these six sections, section 5.7, compares the classification results for seven different values of batch sizes.

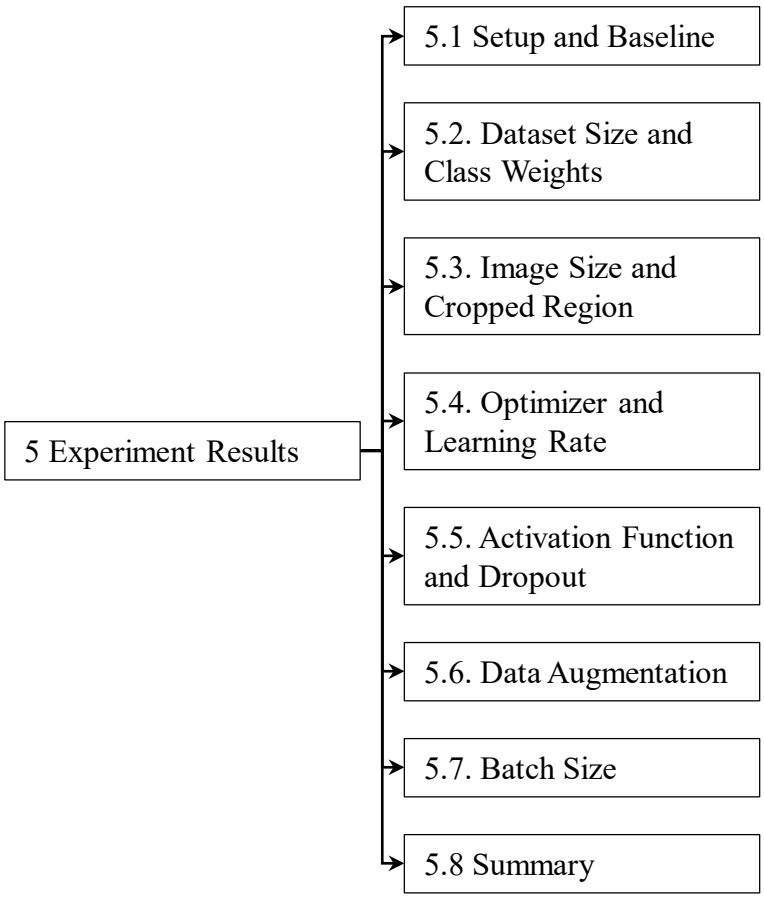


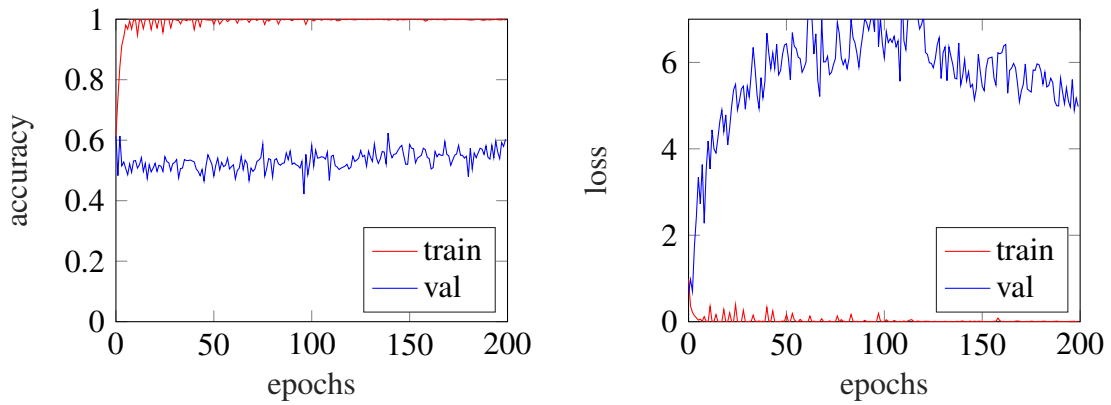
Figure 5.1.: Overview of the sections in this chapter

5.1. Setup and Baseline

The experiments conducted in this chapter are always performed in the same way to make them comparable. Only single parameters or inputs are changed from section to section, indicated by their headlines. Basically, we started using the initial network architecture described in 4.3.1 without any changes. The result of this is displayed and discussed below, to give a first impression on how results are presented in the subsequent sections.

Both training and validation data are inputs of the training function, enabling `Keras` to automatically evaluate the model on the validation data after each single epoch and monitor the performance. With callbacks, introduced in section 4.3.3, training curves can be generated, i.e. training and validation curves for accuracy and loss. In this work they are presented as MATLAB [40] plots, in order to discuss the training behavior of a specific setting. Moreover, an additional evaluation of the validation data at epoch 200, the final epoch, was conducted, in order to obtain the confusion matrix, as well as, precision, recall, and f1-score. With this it is easy to see how different trainings perform after the same amount of epochs and which trainings can be discarded. For the models that were used, i.e. presented in this work, the same additional evaluation was conducted on the test data to make sure the network has not seen this data before and thus endorse the obtained results. These results are always referred to as test results in the following. In the majority of cases, the test results are slightly higher (2-4%) than the validation results.

Figure 5.2 shows the accuracy and loss curves of the baseline training. Here, the initial network architecture of fig. 4.2 is used with default parameter settings and the original dataset without class weights. The input images are 300x300 pixel sized crops from the center of the original image. The training ran for 200 epochs with a batch size of 128 and RMSprop optimizer with a constant learning rate of 0.001.



(a) Accuracy for training and validation

(b) Loss for training and validation

Figure 5.2.: Training curves with the original network architecture and default settings, a validation accuracy of 60.46% was obtained at epoch 200

The plots show, that the training accuracy reaches about 100% very fast in the first few epochs and then almost stays constant. The validation accuracy only reaches about 55% which means the model suffers from very heavy overfitting. A possible reason is, that the dataset is too small. The training loss stays almost constant at 0, which corresponds to the

high training accuracy. The validation loss, on the other hand, increases over the first 100 epochs and then slowly decreases again. However, the decrease of the validation loss has no positive influence on the validation accuracy.

Table 5.1 shows the confusion matrix of the model at epoch 200 on the test set. Here an accuracy of 70.66% was reached with a precision, recall and f1-score of 0.51 for cancer cases and 0.79 for healthy cases. These results mark the baseline of the experiments conducted in this work, from which on adaptations are made to achieve improvements in performance.

Table 5.1.: Confusion matrix of the default training at epoch 200

	cancer	healthy
cancer	148 (0.51)	142 (0.49)
healthy	142 (0.21)	536 (0.79)

5.2. Dataset Size and Class Weights

The size of the dataset plays a major role in machine learning. In general, with a lager dataset, the network is less prone to overfitting. However, as mentioned in section 4.2, our dataset is not evenly separated into cancer and healthy cases, this can cause an imbalance in training, as explained in section 3.3.4. There are three options how to use the dataset for training. First, the dataset is directly used for training with no change, second, similar to the first but with a class weight dictionary passed to the training function or third, surplus samples of the healthy class are discarded, in order to equalize the amount of samples per class.

Intuitively option number two is the most reasonable one, as it uses the complete dataset and not a minimized version of it, which is better for the overfitting problem but still takes the imbalance between the classes into account. Therefore, the network does not learn to "prefer" a particular class to gain a better accuracy. Nevertheless, all three options were tested before starting to make any other changes to the network architecture or the input data.

Figure 5.3 shows the comparison of the training and validation accuracy of the three above mentioned options. The validation accuracy develops about the same for all three options and is a lot lower than the training accuracy. At epoch 200, approximately 60% accuracy is achieved on the validation set, whereas on the training set, 100 % were reached already after about 30 epochs. This means, that all three versions of the dataset cause heavy overfitting in the network. However, in the starting phase of the training, little differences for the training accuracy can be spotted. For the version with the class weight dictionary, the curve increases the fastest, followed by the original dataset and then the equalized one. In all following experiments of this work, the original dataset with the class weight dictionary is used. The loss curve can be found in appendix A.4.

5.3. Image Size and Cropped Region

When training the network with several different image sizes it can be seen, that there are severe differences in the performance of these classifiers. Usually, prostate segmentation is

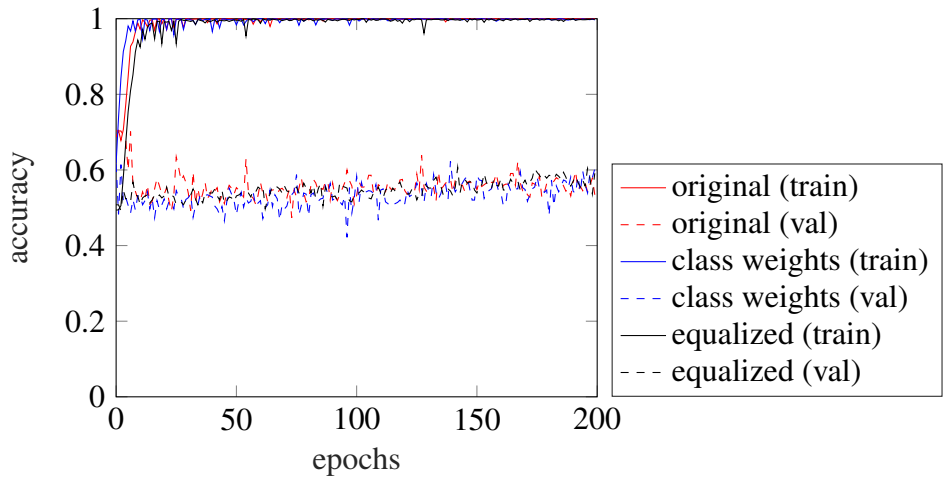


Figure 5.3.: Comparison of accuracy curves for the original dataset size with and without class weights and the dataset with equalized number of samples per class

done in a preprocessing step, however not in this work, as mentioned in chapter 2. Therefore it stands to reason, that a smaller cropped region around the possible cancer area decreases the training time and can increase the potential of correctly detecting the cancer. However, the evaluation results do not show such an improvement. A possible cause can be that our network is used for classification and not segmentation. Therefore surrounding context information is more important for the classifier, than a reduced image snippet would be for correct localization.

Table 5.2 shows the training accuracy for all combinations of the cropped and rescaled sizes of the images. For the rescaling a linear interpolation was used. All images were cropped before training, as it is not possible to use the original sized image. The reason is, that at the edges of the images, especially at the top, there is some patient or setting specific text displayed, which also differs according to the class. Meaning, a different text is displayed for cancer and healthy cases. If a network was trained with this kind of data, the classifier would probably learn to distinguish the displayed text and not the cancer. Therefore, only smaller crops than the original image size are used in this work, such that all the image sizes listed in the table do not include any additional information that could alter the classification result.

Table 5.2.: Test accuracy for different image and crop sizes

resize \ crop	400x400	300x300	200x200	150x150
400x400	0.94	0.69	0.66	0.66
300x300	0.95	0.71	0.68	0.73
200x200	0.94	0.70	0.71	0.69
150x150	0.93	0.67	0.67	0.66

Table 5.3 shows the different training times of the same classifiers as above. The larger the image, the longer the training time, whereas the size of the cropped region does not considerably affect the training time. The reason is simple, the more data there is, i.e. the

larger the input image, the longer the computation time per image, which sums up over epochs. The tables for test precision, recall and f1-score are listed in appendix A.4. All results are of the 200th epoch to keep results comparable.

Table 5.3.: Training times for different image and crop sizes for 200 epochs in hours (x: no representative training times, due to parallel computations)

resize \ crop	400x400	300x300	200x200	150x150
400x400	20.69	20.69	x	x
300x300	11.28	10.99	10.97	x
200x200	5.00	5.00	4.97	4.75
150x150	4.64	4.61	4.59	2.61

5.4. Optimizer and Learning Rate

So far no changes to the initial network architecture have been made, except for the input size. However tuning the hyper-parameters is important to adapt the network and influence the outcome of a training. Therefore this section presents experiment results with three different optimizer types and several learning rates. RMSprop, SGD and Adam optimizers are examined with different constant learning rates, i.e. no decrease during one training process. The dataset with the cropped region of 400x400 pixels, resized to 200x200 pixels has been used, as it achieved one of the highest test accuracies with a relatively short training time (cf. tab. 5.2 and 5.3).

RMSprop Figure 5.4 depicts the accuracy curves of five trainings with RMSprop optimizer with learning rates from 10^{-1} to 10^{-5} with a step size of 10^{-1}. The red and cyan curve show that learning rates of 10^{-1} and 10^{-2} are too high, which causes an oscillation around a local minimum, as mentioned in section 3.2.2. The best and fastest results are obtained with learning rates of 10^{-3} and 10^{-4} in black and blue, whereas the green curve shows that with a training rate of 10^{-5} it takes longer to converge and even drops lower than 50% in the first few epochs. In general, the lower the learning rate, the longer it takes to reach the global minimum. Furthermore, the problem of overfitting can be observed, as well as for SGD and Adam. The loss curves are illustrated in appendix A.4.

SGD Figure 5.5 shows the accuracy curves for trainings with several constant learning rates with SGD optimizer. Here the effect of the learning rate can be observed more distinct than with RMSprop. At epoch 200, training and validation accuracy with learning rates of 10^{-4} and 10^{-5} did not even start to converge yet. At a learning rate of 10^{-3} the validation already converged, as the training accuracy is about to start to do so. The curves of the two higher learning rates converge after a few epochs. The corresponding loss curves are shown in appendix A.4.

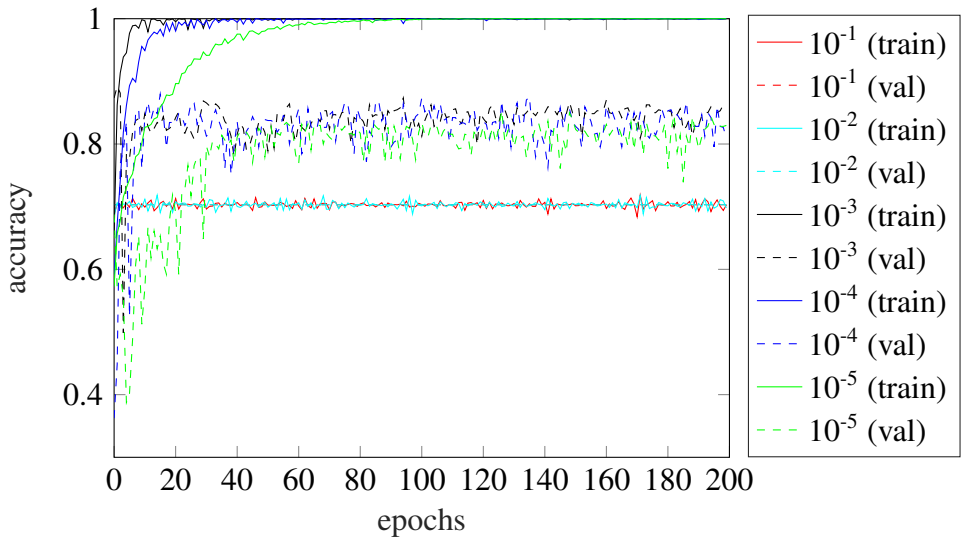


Figure 5.4.: Comparison of accuracy curves for training with different constant learning rates with RMSprop optimizer

Adam Figure 5.6 depicts the accuracy curves for varying learning rates with Adam optimizer (loss curves in appendix A.4). Similar to RMSprop the high learning rates result in an accuracy around 70% for both training and validation that stays constant over all epochs. Also the trainings with the other three learning rates behave similar to RMSprop, the lower the learning rate, the slower the convergence of the accuracy.

Table 5.4 shows the test accuracy at epoch 200 for the trainings presented above. It has to be mentioned, that the accuracy for the lower learning rates with SGD optimizer are probably going to further increase, if the training is continued. `Keras` has default values for the chosen optimizer types, which are also the recommended values: RMSprop - 0.001, SGD - 0.01, and Adam - 0.001. The table shows that these values are the ones that reach the best accuracy results (except for SGD where a learning rate with 0.1 gets 2% more than with 0.01, but this value lays within the range of variation around the converged value). Appendix A.4 includes precision, recall and f1-score results for this test series. The initial network architecture with RMSprop optimizer and a learning rate of 10^{-3} still obtained the best results, which is the reason why these settings will be used in the following sections as well.

Table 5.4.: Test accuracy for three different optimizers with several constant learning rates at epoch 200

	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
RMSprop	0.70	0.70	0.97	0.91	0.90
SGD	0.94	0.92	0.89	0.77	0.63
Adam	0.70	0.70	0.95	0.89	0.91

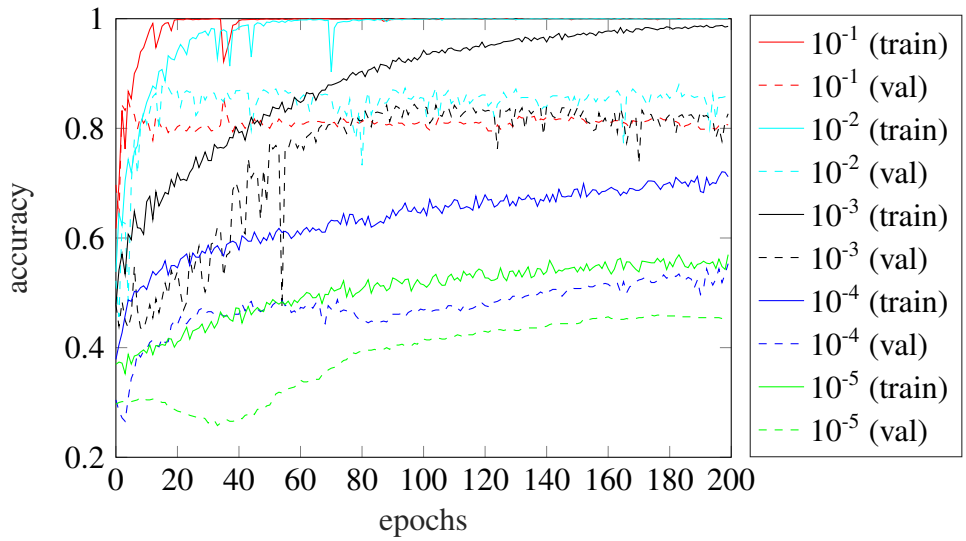


Figure 5.5.: Comparison of accuracy curves for training with different constant learning rates with SGD optimizer

5.5. Activation Function and Dropout

Figure 5.7 shows an extract of the last layers of the network architecture (c.f. fig. 4.2) and the location where this section's changes are applied to. By increasing the dropout, it is generally possible to reduce overfitting, as mentioned in section 3.3.2. Therefore, this section investigates the effect of varying dropout values to our network. Another important aspect is the activation function. In the convolutional layers ReLU activations are used. However, their limitation is, that it should only be used within the hidden layers of a NN. Therefore, the output layer of our initial network architecture uses a softmax activation instead. Also sigmoid, for example, is a reasonable choice for an activation function for binary classification.

Figure 5.8 shows what happens when a ReLU function is selected as the output activation. The training process shows unreasonable behavior and achieves an accuracy of approximately 30% for all dropout values, which is an even worst result than simply "guessing" the classes.

Table 5.5 gives on overview over the test accuracies at epoch 200 of this test series. The columns are for the different dropout values from 0.3 to 0.9 with a step size of 0.1. The rows list three different activation functions, sigmoid, softplus, and softmax (c.f. section 3.2.1). In this case, the type of activation function does not considerably influence the performance or training behavior of the network, as all three functions are quite similar. The different dropout values, however, have an influence on the how fast the network learns or how strong the overfitting is. A more detailed description is given along with the following figures. The only exception in this test series is a training with softplus activation and a dropout of 0.3. Inexplicably, it achieves a constant validation accuracy of 0.70 over all epochs.

Figure 5.9 illustrates the accuracy training curves with sigmoid activation in the output layer and several different values for the dropout. It can be observed, that the training converges slower with increasing dropout value. As in the dark blue curve, for a dropout of 0.9, the validation accuracy is higher than the training accuracy in the first few epochs. This is a

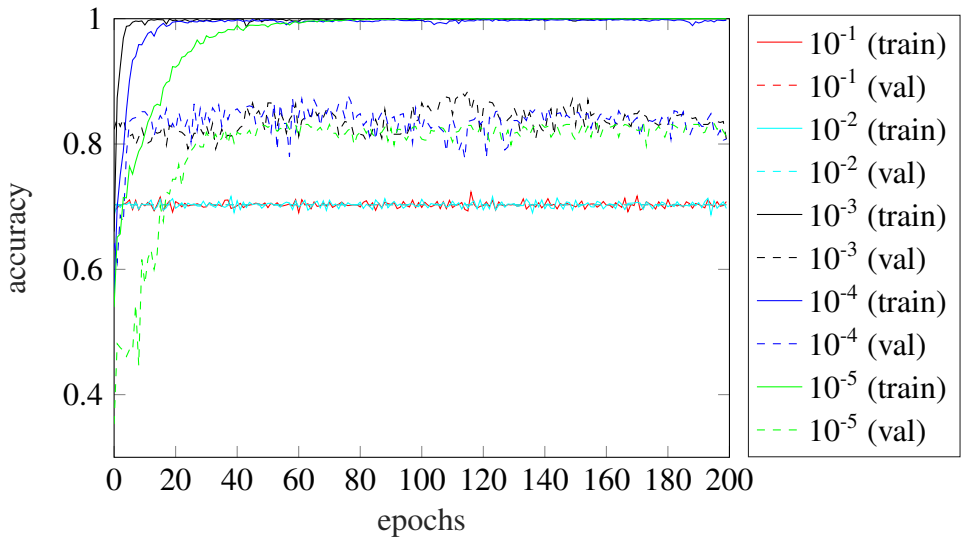


Figure 5.6.: Comparison of accuracy curves for training with different constant learning rates with Adam optimizer

Table 5.5.: Test accuracy for different activation functions of the output layer and dropout values at epoch 200

activation \ dropout	0.3	0.4	0.5	0.6	0.7	0.8	0.9
sigmoid	0.96	0.94	0.95	0.95	0.96	0.91	0.93
softplus	0.70	0.92	0.92	0.95	0.95	0.93	0.94
softmax	0.92	0.94	0.95	0.96	0.93	0.96	0.94

typical effect that can occur when using dropout in a NN since the behavior in training and validation differs. In training, a percentage of the features are set to 0 (here 50%, as we use a dropout of 0.5). When validating, however, all features are used and scaled appropriately, so the model is more robust at the validation time, which can lead to higher validation accuracies. Another observation in this diagram is, that the gap between the training and the validation accuracy decreases with increasing dropout value, i.e. less overfitting. In case of the blue curve (dropout 0.9) and also the magenta (0.8) and green (0.7) one, the training accuracy did not converge yet at epoch 200. Therefore, it can be said, that there is more potential for further improvement than for the other curves, even though the validation accuracy seem to have converged already.

Figure 5.10 and 5.11 show similar results to fig. 5.9, except for the previously mentioned aberration for a dropout of 0.3 with softplus activation.

5.6. Data Augmentation

Data augmentation is a useful tool in many image classification tasks. It allows artificial enlargement of the dataset, which means generation of more training samples, thus a better

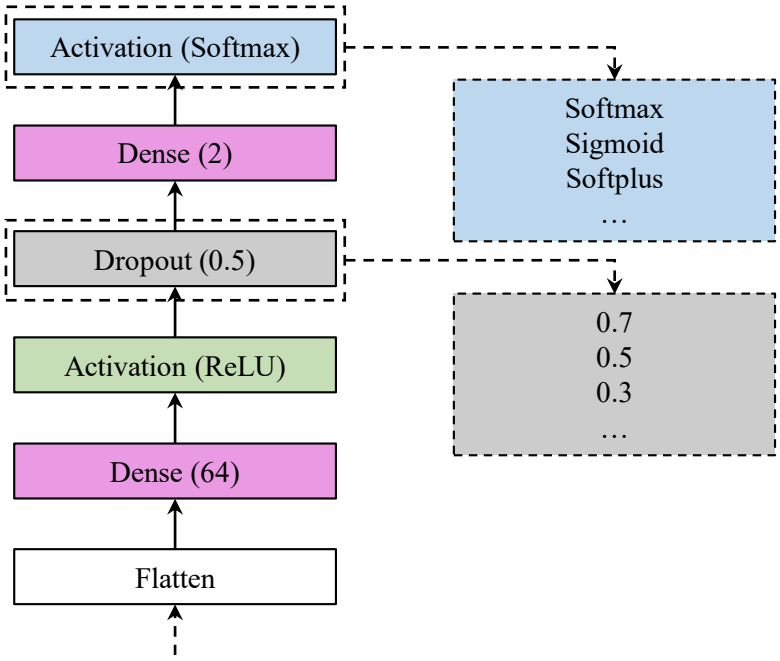


Figure 5.7.: Extract of the fully connected layers of the current network architecture (cf. fig. 4.2) shown with some applicable options to change the output activation function and dropout.

chance for a good classification result with reduced overfitting. Simple data augmentation techniques usually are cropping, rotating, and flipping input images. For many objects, such as animals or vehicles, this is a good way to generate more training samples, as those objects can occur in several positions and rotations in an image. However, in medical applications this is usually not the case, because scans are more homogeneous. In most cases, the object of interest is located in the center of the image and is aligned in a certain direction, furthermore the color spectrum is normalized in most cases. Therefore, augmentations such as mirroring or adding values to specific color channels is unnecessary and would even create unrealistic data, that never appears as an input.

However, the images of the dataset do not look entirely the same (c.f. fig. 4.1), which means, that minimal data augmentation could still help to reduce overfitting. This section compares two trainings with cropped regions of 400x400 pixels, rescaled to 200x200 pixels, and RMSprop optimizer with a constant learning rate of 0.001. One training uses data augmentation, the other does not. The augmented images are generated with the `keras.preprocessing.image.ImageDataGenerator` class, according to the code displayed in section 4.3.1.

The augmentations applied here are a 10 degree range of random rotations, and a 0.1 fraction of the total width and height, as a range for random horizontal and vertical shifts, respectively. Furthermore a 0.1 shear intensity, the shear angle is in counter-clockwise direction as radians, and a 0.1 range for random zoom. Both trainings include a rescale factor of $\frac{1}{255}$ for images in the range of [0, 1), as mentioned in section 4.3.1. The range values are intentionally small, in order to generate realistic augmented data and not exaggerate.

Figure 5.12 shows some examples of augmentations (right side) from the original training image (left side). In this case the `ImageDataGenerator` can generate up to 192 augmented

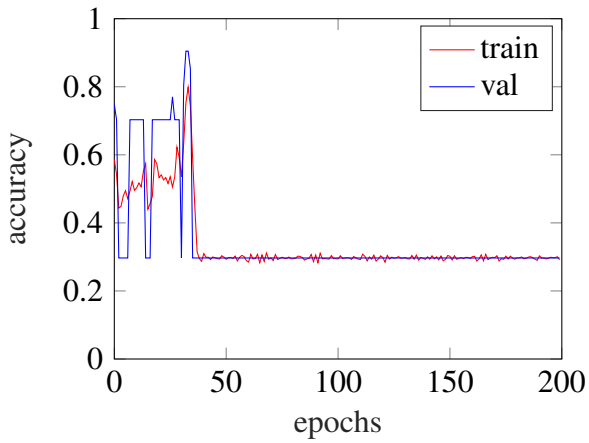


Figure 5.8.: Example of an accuracy curve for an unreasonable training behavior when using a ReLU activation function in the output layer.

samples from one image. The data generator makes sure, the model does not see the exact same picture twice.

Figure 5.13 and 5.14 show the curves of the training accuracy and loss, respectively. Table 5.6 and 5.7 show the confusion matrix of the validation at epoch 200 for training without and with data augmentation, respectively. First, it can be observed that the validation results are lower when using data augmentation, which leads us to the conclusion that data augmentation is not suitable for this application. On the test dataset an accuracy of 83.99% is achieved for data augmentation at epoch 200, without augmentation accuracy is 93.70%. However, looking at the end result is not sufficient. Taking a look at the training curves, reveals that the training behavior differs for the two options. Training accuracy increases slower with data augmentation and is only about to converge at epoch 200, furthermore the loss is lower than when using no data augmentation. However, the validation accuracy is higher than the training accuracy in the beginning of the training, which is attributable to the usage of dropout, as already mentioned in the previous section. In average the validation accuracy for training with data augmentation is slightly higher then the one without, but also shows higher fluctuations from epoch to epoch. However, overfitting is not as strong here as without data augmentation, which can be seen by the reduced offset between training and validation accuracy.

Table 5.6.: Confusion matrix of training without data augmentation at epoch 200

	cancer	healthy
cancer	253 (0.87)	37 (0.13)
healthy	24 (0.04)	654 (0.96)

Table 5.7.: Confusion matrix of training with data augmentation at epoch 200

	cancer	healthy
cancer	207 (0.71)	83 (0.29)
healthy	72 (0.11)	606 (0.89)

It can be said, that in our case, data augmentation is capable of reducing the problem of overfitting, but the architecture is not able to reach the maximum training accuracy anymore (at least not within 200 epochs, longer training could change that) and thus does not generate

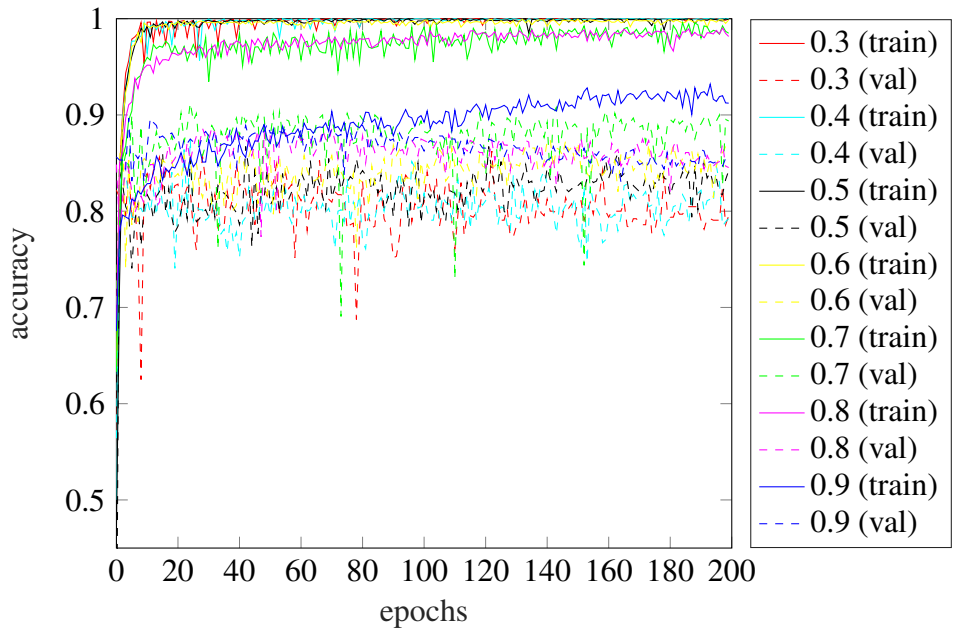


Figure 5.9.: Comparison of accuracy curves for training with different dropout values and sigmoid activation function in the output layer

any better results on the test dataset. However, it gives the possibility to investigate further adaptations of the network architecture to achieve improvements of the performance.

5.7. Batch Size

The batch size usually does not have a big influence on the final outcome of the network performance when trained on enough epochs. But it can influence the training process itself. In general, the smaller the batch, the less accurate the estimate of the gradient. Typically, networks train faster with mini-batches, as weights are updated after each propagation. However, the training times in this section for 200 epochs, are approximately 5.2 hours, independent on the batch size. Again, the initial network architecture was used with 400x400 cropped images, resized to 200x200.

Figure 5.15 shows the accuracy training curves for seven values of batch sizes from 1 to 256. As expected, there is no considerable difference in the test and validation accuracy between the trainings. All range between 80% and 85% and suffer from overfitting. For a batch size of 256, the training accuracy increases a bit slower than for smaller batch sizes, but the effect is not very strong. Appendix A.4 contains the corresponding loss curves. The loss increases over epochs, which is attributable to the problem of overfitting.

5.8. Summary

The overall conclusion, that can be drawn, is that all the trainings in this work suffer from overfitting. The reason is not related to a too big or complex network architecture, as the

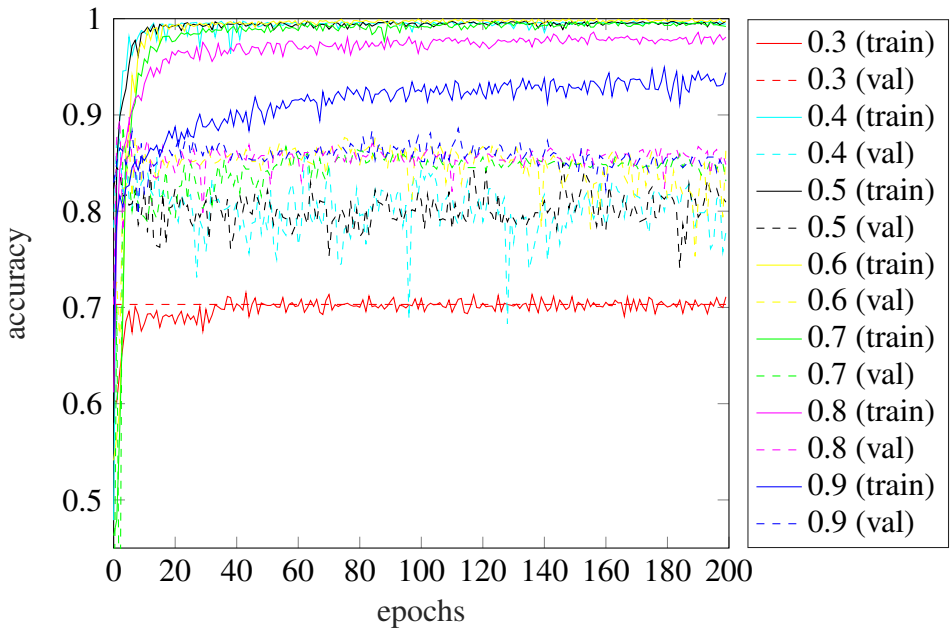


Figure 5.10.: Comparison of accuracy curves for training with different dropout values and softplus activation function in the output layer

one used here is quite simple. The problem rather corresponds to a too small dataset, which can be seen when using data augmentation. Here, the gap between training and validation accuracy decreased. The size of the cropped region from the image influences the outcome of the training, this and the mentioned dataset size, show that the input data is a very important factor for the training result. In the future a larger dataset and more extensive data preprocessing could help improve the performance. However, most of the changes applied to the network itself, in the above mentioned experiments, did not considerably increase the performance. Therefore, prospectively, the focus should be on extending and improving the architecture and not to further tune the hyper-parameters.

Additional training results are presented in appendix A.5. In this work, further experiments with testing of the VGG16 model [35] and `Keras`-RetinaNet [41] for object localization were conducted. VGG16 did not perform better on the dataset than our network, only the training time was about ten times longer. However a deeper investigation of both experiments would exceed the extent of this thesis, thus is not further described. However, in the future, some modifications might be beneficial to the result, e.g. use the `LearningRateScheduler` (section 4.3.3) during training, set the class decision threshold to a higher value than 50%, or to include only one image per patient, but only if the general dataset is bigger. To decrease training times, it might be helpful to use only one channel images, as they still include all the necessary information.

The most important assumption from these test results is, that additional information about the object of interest, i.e. the tumor, is needed for a better classification. So far we do not tell the network what it shall pay attention to, this can make it more difficult for the network to achieve a good result. Adding the information about the localization of the object, explicitly shows the network which and where the object of interest is and could help it train in a more specific way. However, this action automatically indicates to implement the next step, which

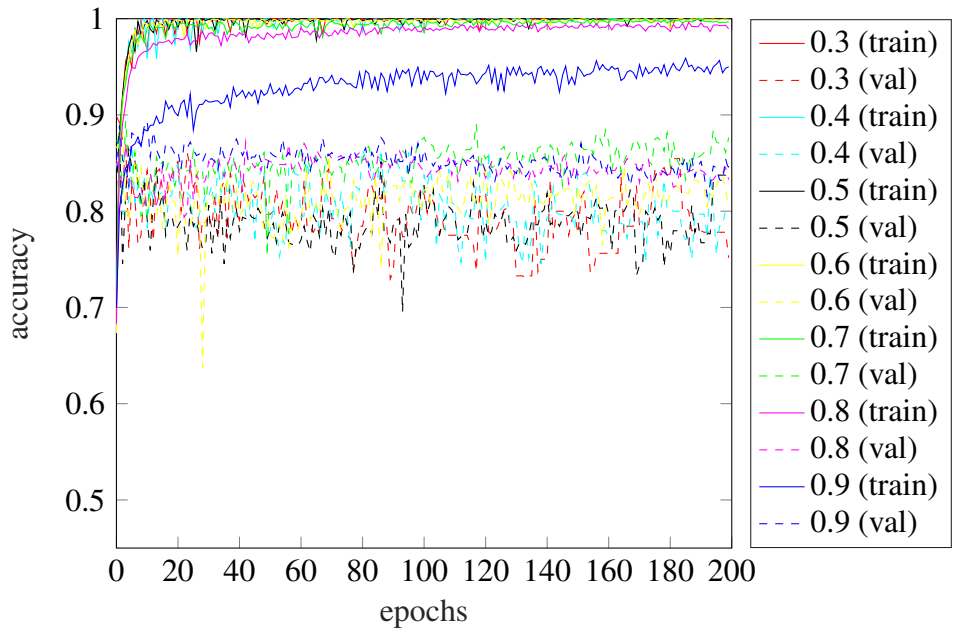


Figure 5.11.: Comparison of accuracy curves for training with different dropout values and softmax activation function in the output layer

is the localization/segmentation of the tumor.

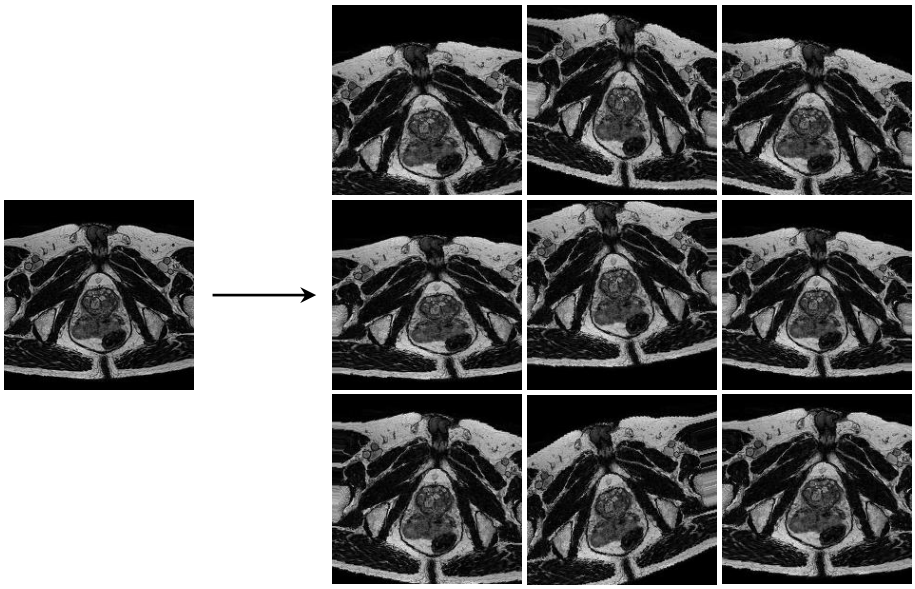


Figure 5.12.: When data augmentation is applied to an original image of the dataset (left side), several augmented versions (up to 192 in this case) of the image are generated. Here a selection of nine examples is shown (right side).

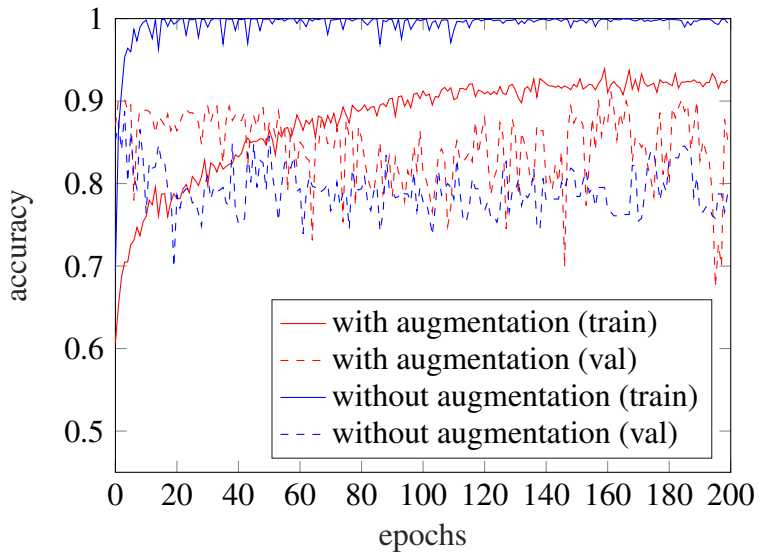


Figure 5.13.: Comparison of accuracy curves for training with and without data augmentation

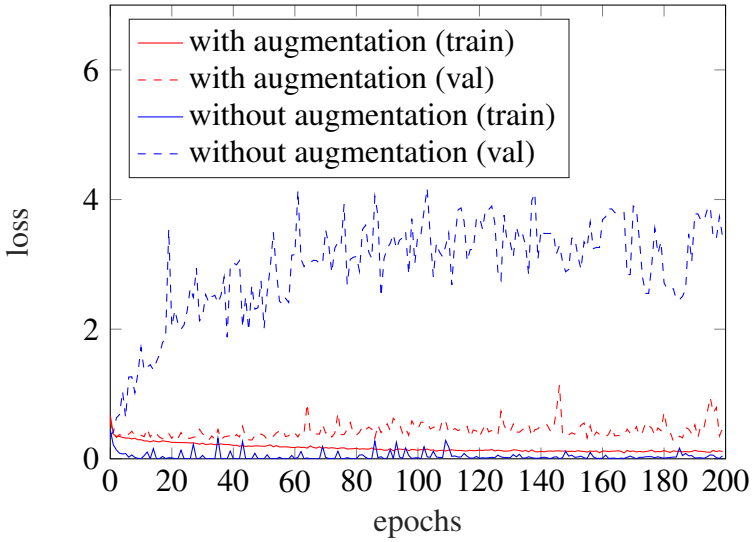


Figure 5.14.: Comparison of loss curves for training with and without data augmentation

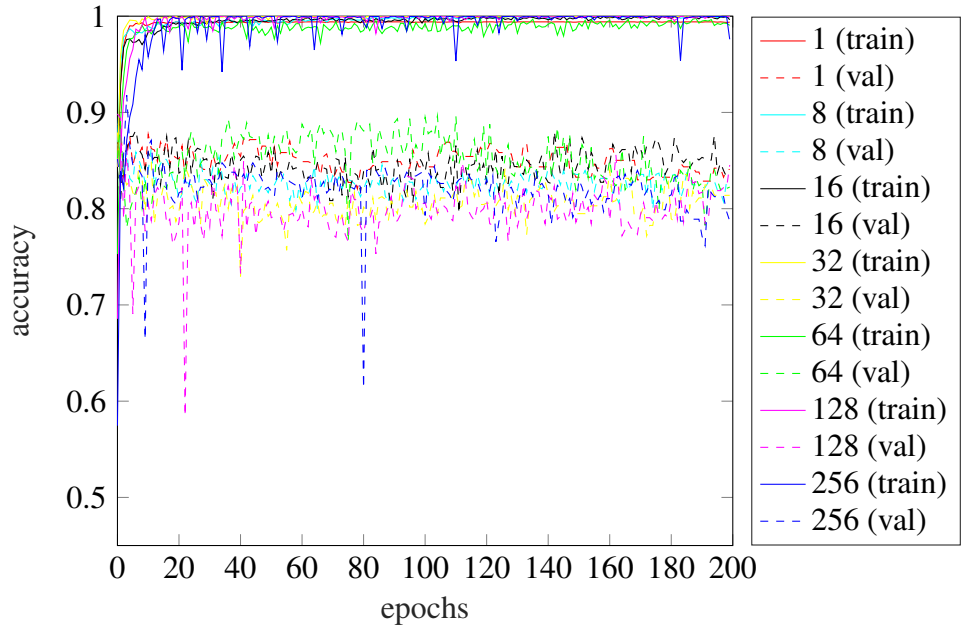


Figure 5.15.: Comparison of accuracy curves for training with different batch sizes

6. Conclusion and Outlook

Training a neural network is a complex task that requires a lot of experience, testing, evaluating and decision making. It can be very time consuming to adapt a model for a new application problem and there may be many failures. This work shows that the choice of the parameter settings can have a huge impact on the overall performance of a model. As for example, selecting a ReLU activation in the output layer, causes an unpredictable training behavior, but when a correct activation function is chosen, such as softmax, the accuracy on the test data exceeds 90%.

However, it was observed that the influence of many of the tested hyper-parameters is not as strong as anticipated to reduce overfitting. The input data, on the other hand, plays a major role in this matter. The size of the cropped region heavily influences the results in terms of accuracy. Furthermore, the rescaled size of this crop determines the training time, but not the performance. The effect of data augmentation on the training process, is to enable reduction of overfitting but it also reduces the general training accuracy. However, the augmentations show potential of improving the classification result in the future. The relatively small size of the dataset seems to be the main limitation of this work. Further testing is required to show if the network yields better results with a larger dataset.

Depending on the type of optimizer and the learning rate chosen, training accuracies either increase faster, slower, or even get stuck in a local minimum, not improving at all. The RMSprop optimizer with a learning rate of 0.001 achieves the highest accuracy on the test set with 97% at epoch 200. For future experiments it might be beneficial to use an adaptive learning rate, which decreases over epochs to improve the training behavior. Batch size and dropout do not seem to have such big influence on the classification result, and they might only be used for fine-tuning in the later adaptation stages.

For the future, an essential step is to include the localization information of the cancer. This information may help the model obtain better accuracies. Currently, the network does not get a "hint" where it should pay attention to. This could cause the network to take into account certain features, that are not directly related to the cancer itself. By including localization information, such as bounding boxes or better heat maps, this can be avoided. This marks the beginning of the next step towards implementing the segmentation of the cancer itself. It is also useful for future experiments to conduct a prostate segmentation prior to the cancer recognition task. This requires the dataset to contain these segmentation information.

In general, this work includes the training and presentation of over 100 models, excluding the number of test and failed trainings at the beginning of the implementation phase. We achieved an improvement of 37% from the initially proposed network architecture and settings to a tuned network with 97% accuracy on the test set. This high score enables a trustworthy classification of MRI scans for prostate cancer detection and therefore helps doctors in their decision making for diagnosis.

A. Appendix

A.1. Anatomy of the Prostate

The prostate (or prostate gland) is part of the male reproductive and urinary system in most mammals. It differs considerably among species anatomically, chemically, and physiologically. In young men it is about the size of a walnut, but it enlarges when men reach their late 40s and early 50s.

Figure A.1 shows that the prostate is located deep inside a men's pelvis, below the bladder and in front of the rectum. It wraps around the upper part of the urethra, which is a tube that carries urine from the bladder, through the prostate and penis out of the body.

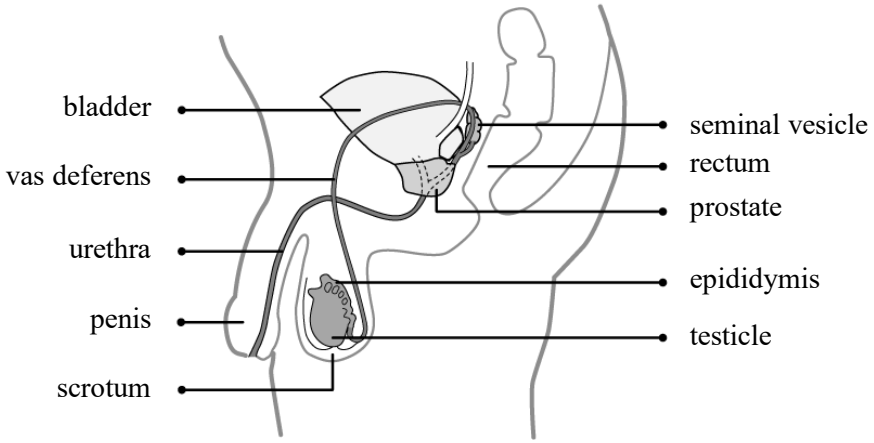


Figure A.1.: The male reproductive system (source: [42])

Figure A.2 shows that the prostate gland is composed of different types of tissue, divided into zones [43]:

- The peripheral zone (PZ) contains the majority of prostatic glandular tissue. The largest area of the PZ is at the back of the gland, closest to the rectal wall. When a doctor performs a digital rectal examination it is the back surface of the gland he/she is feeling. About 70-80% of prostate cancers originate in the peripheral zone.
- The central zone (CZ) is the area that surrounds the ejaculatory ducts. Only a very small percentage of prostate cancers begin here (less than 5%) and are thought to be more aggressive and more likely to invade the seminal vesicles.
- The transition zone (TZ) surrounds the urethra as it enters the prostate gland. It is small in young adults, but it grows throughout life, taking up a bigger percentage of the gland, and is responsible for benign prostatic hyperplasia (BPH). Roughly 20% of prostate cancers begin in this zone.

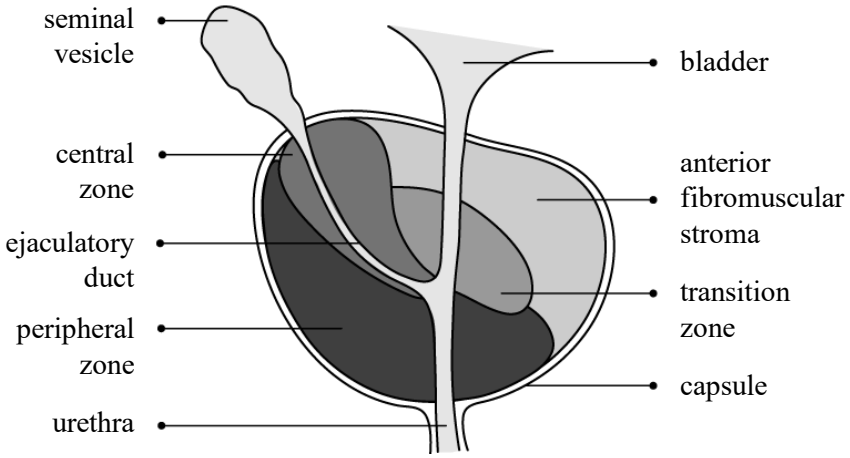


Figure A.2.: Zones of the prostate (source: [42])

Unlike ordinary ultrasound, mpMRIs is able to depict the three zones of the gland and to differentiate between healthy versus diseased tissue. In addition, mpMRIs can detect significant prostate cancer in any of the zones, and show existing extracapsular extension (tumor at or beyond the edge of the prostate capsule).

A.2. Introduction to Magnetic Resonance Imaging (MRI)

This section gives a short explanation about Magnetic resonance imaging (MRI) and is based on [44] and [45].

MRI is an imaging technique used primarily in medical settings (radiology) to produce high quality images of the inner anatomy and physiological processes of the human body. It is based on the principles of nuclear magnetic resonance (NMR), which is a spectroscopic technique used to obtain microscopic chemical and physical information about molecules. MRI scanners use strong magnetic fields, electric field gradients, and radio waves to generate images of thin slices through the human body.

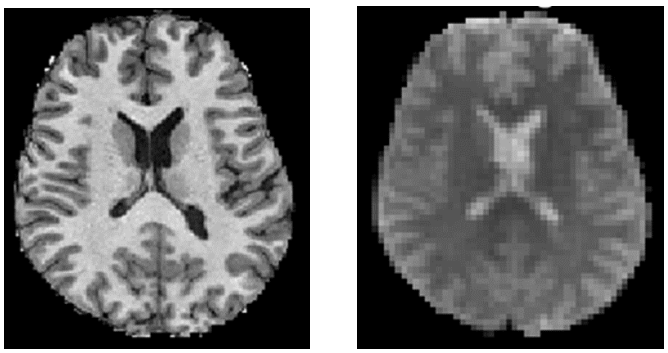
In 1946 Felix Bloch and Edward Purcell, independently, discovered the magnetic resonance phenomenon, which was later used to develop the NMR for chemical and physical molecule analysis. In the 1970s, Raymond Damadian found out, that nuclear magnetic relaxation times differ for tissues and tumors, which marked the beginning of detecting diseases with this medical imaging hardware. The basis of current MRI techniques was proposed by Richard Ernst in 1975, who used phase and frequency encoding, and the Fourier Transform for MRI. Two years later, Peter Mansfield developed the echo-planar imaging (EPI) technique, which was later used to produce images at video rates (30 ms/image). By 1986, imaging time was reduced to about five seconds and one year later EPI was used to perform real-time movie imaging of a single cardiac cycle. The functional MRI (fMRI) was developed in 1992, which allows mapping of the function of various regions of the human brain and opened up a new application for EPI.

MRI is based on the absorption and emission of energy in the radio frequency range of the electromagnetic spectrum. Basically, it is a tomographic imaging modality for producing

NMR images of a slice through the human body. Each slice has a thickness and is composed of voxels, their volume is approximately $2mm^3$. The intensity of the pixels in the MR-image is proportional to the NMR signal intensity of the contents of the corresponding voxel of the imaged object.

To obtain these scans, the patient is positioned within the MRI scanner that builds up a strong magnetic field around the area of interest. A human body preliminary consists of fat and water which are composed of many hydrogen atoms (make approx. 63% of the human body). Their nuclei have an NMR signal which creates the MR-image. First, energy of the oscillating magnetic field is applied to the patient. The excited hydrogen atoms emit a radio frequency signal, which is then measures by a receiving coil. Thereby, several types of images can be generated.

Figure A.3 shows two of such types, T1- and T2-weighted images. Each tissue returns to its equilibrium state after excitation by the relaxation processes. For T1, it is a spin-lattice relaxation, i.e. magnetization in the same direction as the static magnetic field. T2 is a spin-spin relaxation, which traverses to the static magnetic field. Depending on the application it is better to use the one or the other method. E.g. T1 is rather used for obtaining morphological information, identifying fatty tissue, characterizing focal liver lesions, as well as for post-contrast imaging. Detecting edema and inflammation, reveling white matter lesions and assessing zonal anatomy in the prostate and uterus is rather done with T2-weighted images.



(a) T1-weighted (b) T2-weighted

Figure A.3.: Examples of T1- and T2-weighted images of an MRI scan of a human head, illustrating their brain. (source. [46])

A.3. Technical Details

Table A.1.: A summary of the technical details of the GeForce GTX 1070, more information can be found on the NVIDIA homepage [47]

	version
Architecture	Pascal
Cores	1920
Performance	3X
Frame Buffer	8 GB GDDR5
Memory Type-Speed	8 Gbps
Boost Clock	1.4x (relative) 1683 MHz (actual)

A.4. Additional Experiment Results

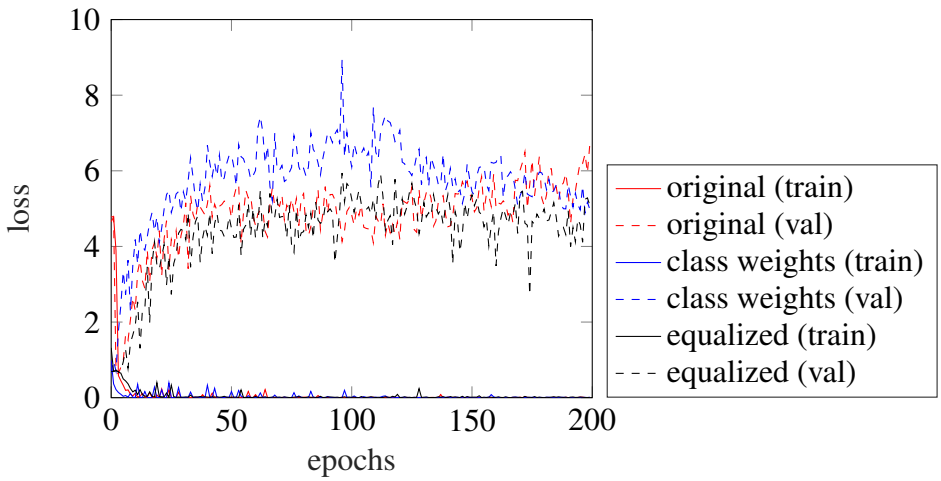


Figure A.4.: Comparison of loss curves for the original dataset size with and without class weights and the dataset with equalized number of samples per class

Table A.2.: Test precision for different image and crop sizes. Numbers in brackets are average/total precision, the other for the cancer class.

resize \ crop	400x400	300x300	200x200	150x150
400x400	0.90 (0.94)	0.48 (0.71)	0.45 (0.70)	0.43 (0.66)
300x300	0.99 (0.95)	0.51 (0.71)	0.47 (0.69)	0.55 (0.72)
200x200	0.91 (0.94)	0.50 (0.73)	0.55 (0.68)	0.48 (0.68)
150x150	0.92 (0.93)	0.47 (0.74)	0.47 (0.70)	0.43 (0.66)

Table A.3.: Test recall for different image and crop sizes. Numbers in brackets are average/-total recall, the other for the cancer class.

resize \ crop	400x400	300x300	200x200	150x150
400x400	0.90 (0.94)	0.59 (0.69)	0.61 (0.66)	0.43 (0.66)
300x300	0.84 (0.95)	0.51 (0.71)	0.51 (0.68)	0.52 (0.73)
200x200	0.87 (0.94)	0.62 (0.70)	0.20 (0.71)	0.41 (0.69)
150x150	0.85 (0.93)	0.75 (0.67)	0.60 (0.67)	0.46 (0.66)

Table A.4.: Test f1-score for different image and crop sizes. Numbers in brackets are average/total f1-score, the other for the cancer class.

resize \ crop	400x400	300x300	200x200	150x150
400x400	0.90 (0.94)	0.53 (0.70)	0.52 (0.67)	0.43 (0.66)
300x300	0.91 (0.95)	0.51 (0.71)	0.49 (0.69)	0.53 (0.73)
200x200	0.89 (0.94)	0.56 (0.71)	0.29 (0.66)	0.44 (0.68)
150x150	0.88 (0.93)	0.58 (0.68)	0.52 (0.68)	0.45 (0.66)

Table A.5.: Test precision for three different optimizers with several constant learning rates at epoch 200. Numbers in brackets are average/total precision, the other for the cancer class.

	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
RMSprop	0.00 (0.49)	0.00 (0.49)	0.99 (0.97)	0.82 (0.91)	0.85 (0.90)
SGD	0.91 (0.94)	0.87 (0.92)	0.84 (0.89)	0.62 (0.77)	0.35 (0.61)
Adam	0.00 (0.49)	0.00 (0.49)	0.89 (0.95)	0.81 (0.90)	0.84 (0.91)

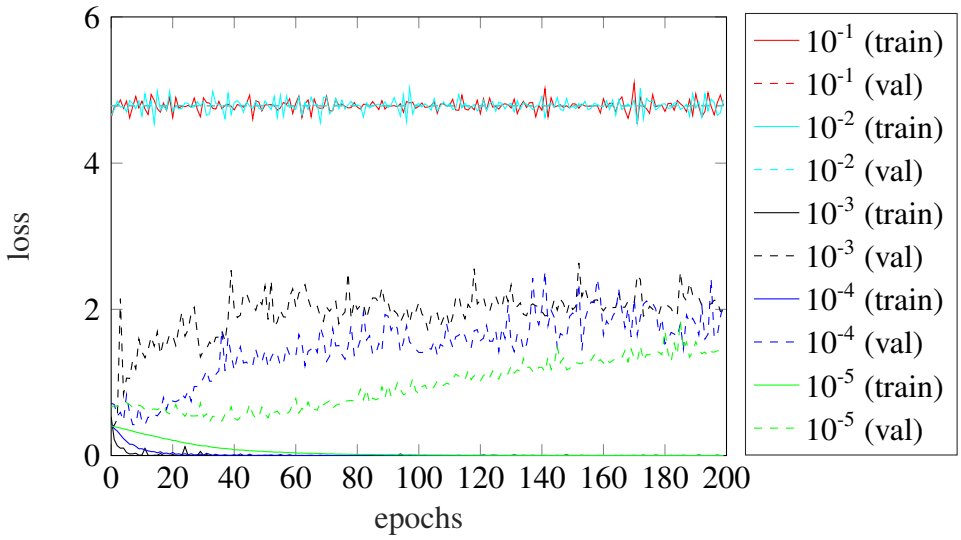


Figure A.5.: Comparison of loss curves for training with different constant learning rates with RMSprop optimizer

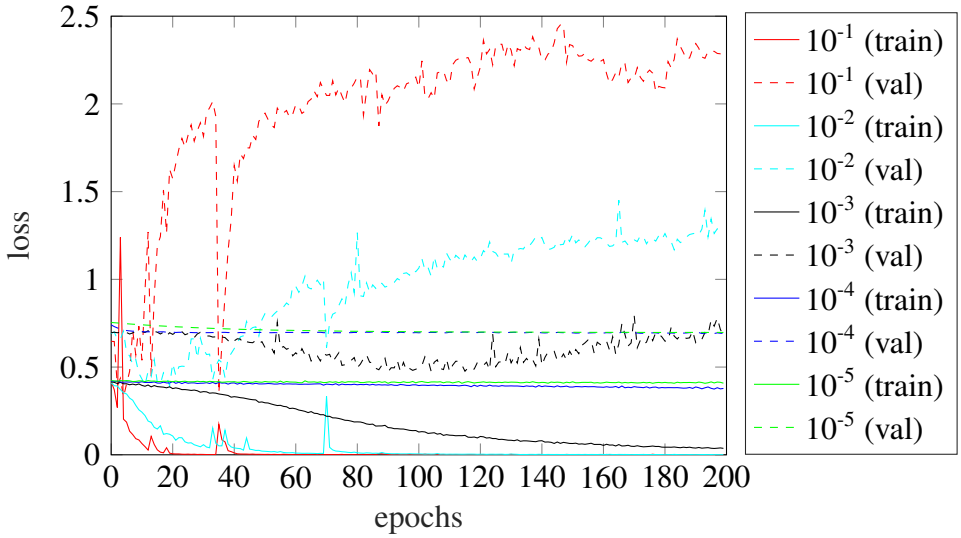


Figure A.6.: Comparison of loss curves for training with different constant learning rates with SGD optimizer

Table A.6.: Test recall for three different optimizers with several constant learning rates at epoch 200. Numbers in brackets are average/total recall, the other for the cancer class.

	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
RMSprop	0.00 (0.70)	0.00 (0.70)	0.91 (0.97)	0.89 (0.91)	0.82 (0.90)
SGD	0.88 (0.94)	0.86 (0.92)	0.79 (0.89)	0.60 (0.77)	0.28 (0.63)
Adam	0.00 (0.70)	0.00 (0.70)	0.96 (0.95)	0.86 (0.89)	0.85 (0.91)

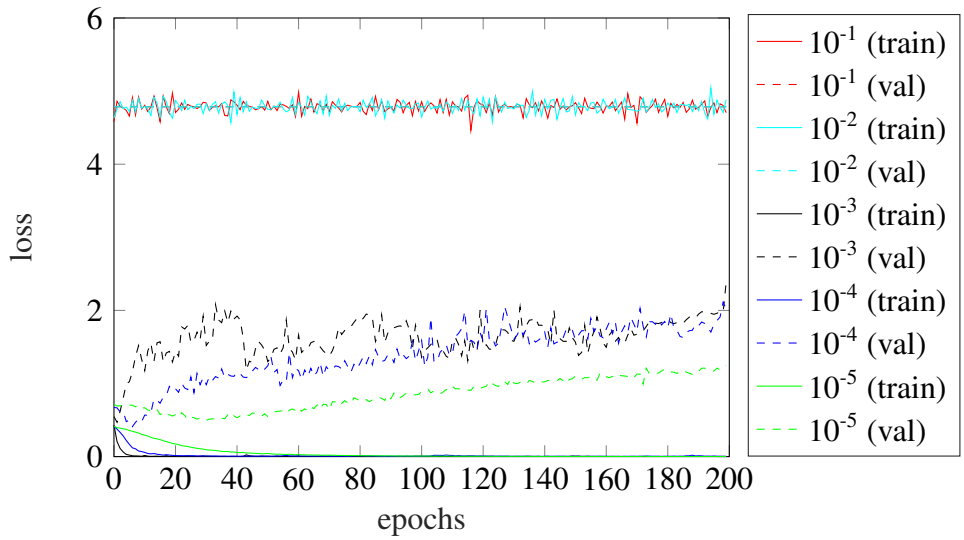


Figure A.7.: Comparison of loss curves for training with different constant learning rates with Adam optimizer

Table A.7.: Test f1-score for three different optimizers with several constant learning rates at epoch 200. Numbers in brackets are average/total f1-score, the other for the cancer class.

	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
RMSprop	0.00 (0.58)	0.00 (0.58)	0.95 (0.97)	0.85 (0.91)	0.84 (0.90)
SGD	0.89 (0.94)	0.86 (0.92)	0.81 (0.89)	0.61 (0.77)	0.31 (0.62)
Adam	0.00 (0.58)	0.00 (0.58)	0.92 (0.95)	0.83 (0.90)	0.85 (0.91)

Table A.8.: Test precision for different activation functions of the output layer and dropout values at epoch 200. Numbers in brackets are average/total precision, the other for the cancer class.

activation \ dropout	0.3	0.4	0.5	0.6	0.7	0.8	0.9
sigmoid	0.96 (0.96)	0.91 (0.94)	0.93 (0.95)	0.92 (0.95)	0.93 (0.96)	0.85 (0.91)	1.00 (0.94)
softplus	0.00 (0.49)	0.85 (0.92)	0.86 (0.92)	0.91 (0.95)	0.93 (0.95)	0.96 (0.93)	0.99 (0.94)
softmax	0.84 (0.92)	0.89 (0.94)	0.94 (0.95)	0.92 (0.96)	0.85 (0.93)	0.97 (0.96)	0.98 (0.94)

Table A.9.: Test recall for different activation functions of the output layer and dropout values at epoch 200. Numbers in brackets are average/total recall, the other for the cancer class.

activation \ dropout	0.3	0.4	0.5	0.6	0.7	0.8	0.9
sigmoid	0.91 (0.96)	0.90 (0.94)	0.89 (0.95)	0.91 (0.95)	0.92 (0.96)	0.86 (0.91)	0.78 (0.93)
softplus	0.00 (0.70)	0.89 (0.92)	0.89 (0.92)	0.92 (0.95)	0.90 (0.95)	0.79 (0.93)	0.79 (0.94)
softmax	0.89 (0.92)	0.92 (0.94)	0.88 (0.95)	0.96 (0.96)	0.93 (0.93)	0.88 (0.96)	0.82 (0.94)

Table A.10.: Test f1-score for different activation functions of the output layer and dropout values at epoch 200. Numbers in brackets are average/total f1-score, the other for the cancer class.

activation \ dropout	0.3	0.4	0.5	0.6	0.7	0.8	0.9
sigmoid	0.93 (0.96)	0.90 (0.94)	0.91 (0.95)	0.92 (0.95)	0.93 (0.96)	0.85 (0.91)	0.87 (0.93)
softplus	0.00 (0.58)	0.87 (0.92)	0.87 (0.92)	0.91 (0.95)	0.91 (0.95)	0.87 (0.93)	0.88 (0.93)
softmax	0.86 (0.92)	0.91 (0.94)	0.91 (0.95)	0.94 (0.96)	0.89 (0.93)	0.92 (0.96)	0.89 (0.94)

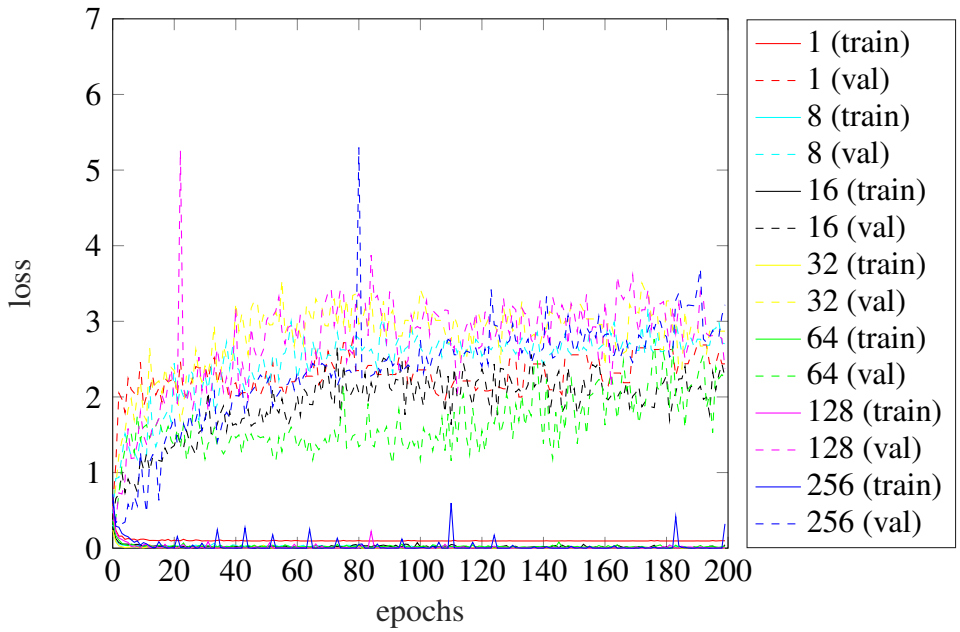


Figure A.8.: Comparison of loss curves for training with different batch sizes

A.5. Failures and additional Experiments

In the beginning of a new research with NNs a lot can go wrong. In most cases, it is easy to see if something is not working correctly. The more difficult part is to find out why and how to fix it. A good indicator for an implementation or other error, is an "awfully good" classification result in one of the first trainings. The probability to achieve an over 95% performance, just by chance, is low.

One of the first trainings in this work achieved such a performance, it was caused by two mistakes in the data handling. Number one, the dataset was split up into training, validation and testing just by image. As there exists more than one image per patient, it occurred that one patient appeared in more than one of the sub-datasets. It can be argued, that this is not too bad, as all the images per patient are still different. However, they are similar enough to massively increase the accuracy. The second error was because the original image data was used. As the images feature some texts, which differs for patient, scan, and class (cancer vs. healthy), the classifier simply learned features from the text instead of the prostate image itself. This was fixed, by using a maximum crop region of 400x400 pixels of the image, to avoid having any text in the input images. After applying those changes reasonable results appeared.

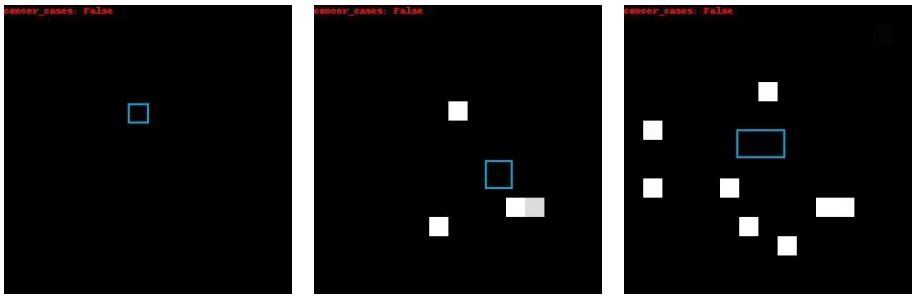
However, it is hard to tell weather the classifier pays attention to object/region of interest (ROI) or classifies by some other information, that correlate to the ROI. In localization/segmentation, the output results better indicate the correctness of the learned task. Not only because it gets more specific information about the object of interest in the training (additional true-false map of the image), but also because of a difference in the validation phase. For example, a model gets an MR-scan as an input and outputs "yes, this contains cancer" versus a model, that gets the same input, but outputs a bounding box or heat map, that shows the location of the cancer. For the latter one, a human can immediately say, that the model correctly recognized the cancer. In the first case, the network simply has to be "trusted" to really recognize the cancer and not some other related feature, that might indicate that this image contains cancer.

Therefore, in many deep learning applications it is useful to visualize the outputs of the hidden layers of a network or to make an additional evaluation at the end, in order to investigate to what the model "pays attention" to. In this work, we created a so-called pseudo-heatmap for our classification results. Which means, random images of the test set are selected, over which our programs moves a sliding window. The sliding window blackens a region of the image (e.g. 20x20 pixels) which is then given to the model to classify. The classification result is being saved for this specific region, then the window moves to the next position. With this we were able to investigate changes in the classification result, when a specific region in the image is covered. In case the window covers a region with the cancer in a cancer image, the classification result should change, as the classifier does not see the cancer anymore. For all other positions, the result should stay approximately the same. Classification result, in this case, means the probability vector, that our network outputs for each class, i.e. [0.37, 0.63] means the model says it is cancer with 37% and healthy with 63%, normally the threshold is 50% to decide for one class.

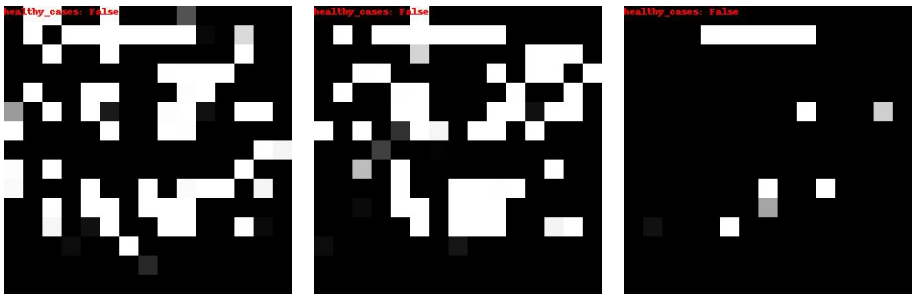
Figure A.9 shows examples for all four possible classification results, true positive (TP) A.9a, false negative (FN) A.9b, false positive (FP) A.9c, and true negative (TN) A.9d. The brighter

a grid cell (generated by the sliding window), the higher the difference of the probability result obtained by the model between the classification of the original image and the image with the current sliding window position. The images of the cancer cases class additionally feature a light blue rectangle, which marks the position of the cancer. In a correct classification scenario for cancer case images, bright regions should appear inside (or at least close to) the blue rectangle, as it can be seen in fig. A.9a. If the image belongs to the healthy class, there should be no change in the classification probability, i.e. the heatmap must be completely black, as shown in fig. A.9d. For false classifications several scenarios can occur. Fig. A.9c shows the case of a false positive classification, i.e. the image belongs to the healthy class, but was classified as cancer. Here many and strong probability changes occur, which also appear randomly scattered over the image. In false negative classifications, either no probability changes occur, as it can be seen in the leftmost picture of fig. A.9b or the probability changes do not occur where the cancer is located (blue rectangle), as shown in the right two images of fig. A.9b. However, with this very easy method, it was possible to confirm, that the network actually classifies the cancer and not something else.

(a) Cancer images correctly classified as cancer (TP)



(b) Cancer images incorrectly classified as healthy (FN)



(c) Healthy images incorrectly classified as cancer (FP)

(d) Healthy images correctly classified as healthy (TN)

Figure A.9.: Examples of the pseudo-heatmap for TP, FN, FP, and TN classifications. The brighter a grid cell, the higher the classification probability change. The blue rectangles in the upper two figures mark the location of the cancer.

List of Acronyms

AdaGrad Adaptive Gradient Algorithm
Adam Adaptive Moment Estimation
ADC apparent diffusion coefficient
AI artificial intelligence
ANN artificial neural network
API application programming interface
AUC area under receiver operating characteristic curve
BN Batch Normalization
BPH benign prostatic hyperplasia
CAD computer-aided detection
CNN convolutional neural network
CPU central processing unit
CUDA® Compute Unified Device Architecture
cuDNN CUDA® Deep Neural Network library
CZ central zone
DCE dynamic contrast-enhanced
DNN deep neural network
DRE digital rectal examination
DWI diffusion-weighted imaging
EPI echo-planar imaging
fMRI functional MRI
FN false negative
FP false positive
GB gigabyte (10^9 bytes = 1 000 000 000 bytes)
GiB gibibyte (2^{30} bytes = 1 073 741 824 bytes; 1 GiB ≈ 1.074 GB)
GPU graphics processing unit
GT ground truth
HED Hollistically nested Edge Detection
LBP Local Binary Pattern

MR magnetic resonance
MRI Magnetic resonance imaging
mpMRI multiparametric MRI
NMR nuclear magnetic resonance
NN neural network
OS operating system
PCD prostate cancer detection
PSA prostate-specific antigen
PZ peripheral zone
ReLU Rectifier Linear Unit
RGB red green blue
RMSprop Root Mean Square Propagation
ROI region of interest
SGD Stochastic Gradient Descent
SVM support vector machine
T2W T2-weighted
TN true negative
TP true positive
TRUS transrectal ultrasound
TZ transition zone

List of Figures

List of Tables

Bibliography

[1] American Cancer Society, "Prostate cancer," [Online; accessed 20-November-2017]. [Online]. Available: https://www.cancer.org/

[2] R. L. Siegel, K. D. Miller and A. Jemal, "Cancer statistics, 2017."

[3] I. Chan, W. Wells, R. V. Mulker, S. Haker, J. Zhang, K. H. Zou, S. E. Maier and C. M. C. Tempany, "Detection of prostate cancer by integration of line-scan diffusion, t2-mapping and t2-weighted magnetic resonance imaging; a multichannel statistical classifier," *Medical Physics*, vol. 30, no. 9, pp. 2390–2398, 2003.

[4] E. Niaf, O. Rouvière, F. Mège-Lechevallier, F. Bratan and C. Lartizien, "Computer-aided diagnosis of prostate cancer in the peripheral zone using multiparametric mri," *Physics in Medicine & Biology*, vol. 57, no. 12, p. 3833, 2012.

[5] R. M. Haralick, "Statistical and structural approaches to texture," *Proceedings of the IEEE*, vol. 67, no. 5, pp. 786–804, May 1979.

[6] S. Wang, K. Burtt, B. Turkbey, P. Choyke and R. Summers, "Computer aided-diagnosis of prostate cancer on multiparametric mri: A technical review of current research," vol. 2014, p. 789561, 12 2014.

[7] J. T. Kwak, S. Xu, B. J. Wood, B. Turkbey, P. Choyke, P. A. Pinto, S. Wang and R. Summers, "Automated prostate cancer detection using t2-weighted and high-b-value diffusion-weighted magnetic resonance imaging," vol. 42, pp. 2368–2378, 04 2015.

[8] C. Qian, L. Wang, A. Yousuf, A. Oto and D. Shen, *In Vivo MRI Based Prostate Cancer Identification with Random Forests and Auto-context Model*. Cham: Springer International Publishing, 2014, pp. 314–322.

[9] Y. Guo, G. Wu, L. A. Commander, S. Szary, V. Jewells, W. Lin and D. Shen, *Segmenting Hippocampus from Infant Brains by Sparse Patch Matching with Deep-Learned Features*. Cham: Springer International Publishing, 2014, pp. 308–315.

[10] H.-I. Suk and D. Shen, *Deep Learning-Based Feature Representation for AD/MCI Classification*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 583–590.

[11] Y. K. Tsehay, N. S. Lay, H. R. Roth, X. Wang, J. T. Kwak, B. I. Turkbey, P. A. Pinto, B. J. Wood and R. M. Summers, "Convolutional neural network based deep-learning architecture for prostate cancer detection on multiparametric magnetic resonance images," pp. 10 134 – 10 134 – 11, 2017.

[12] S. Xie and Z. Tu, "Holistically-nested edge detection," in *"Proceedings of IEEE International Conference on Computer Vision"*, 2015.

[13] D. Kriesel, *Ein kleiner Überblick über Neuronale Netze*, 2007. [Online]. Available: availableonhttp://www.dkriesel.com

[14] A. Ng, "Coursera: Machine learning by stanford university," [Online; accessed 14-November-2017]. [Online]. Available: https://www.coursera.org/learn/machine-learning

[15] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, pp. 65–386, 1958.

[16] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, July 1959.

[17] Wikipedia, the free encyclopedia, "Neurons," 2012, [Online; accessed November 17, 2017]. [Online]. Available: https://commons.wikimedia.org/wiki/File%3ANeurons_uni_bi_multi_pseudouni.svg

[18] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, J. Fürnkranz and T. Joachims, Eds. Omnipress, 2010, pp. 807–814.

[19] P. Ramachandran, B. Zoph and Q. V. Le, "Searching for activation functions," *CoRR*, vol. abs/1710.05941, 2017.

[20] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[21] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau and R. Garcia, "Incorporating second-order functional knowledge for better option pricing." pp. 472–478, 01 2000.

[22] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1," D. E. Rumelhart, J. L. McClelland and C. PDP Research Group, Eds. Cambridge, MA, USA: MIT Press, 1986, ch. Learning Internal Representations by Error Propagation, pp. 318–362.

[23] L. Bottou, *Large-Scale Machine Learning with Stochastic Gradient Descent*. Heidelberg: Physica-Verlag HD, 2010, pp. 177–186.

[24] J. C. Duchi, E. Hazan and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," vol. 12, pp. 2121–2159, 07 2011.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[26] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of Physiology*, vol. 160, no. 1, pp. 106–154, 1962.

[27] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[28] CS231n, "Convolutional neural networks for visual recognition," [Online; accessed December, 01 2017]. [Online]. Available: http://cs231n.github.io/convolutional-networks/

[29] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[30] H. Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function," vol. 90, pp. 227–244, 10 2000.

[31] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012.

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

[33] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," vol. 6, pp. 107–116, 04 1998.

[34] F. Chollet *et al.*, "Keras," 2015. [Online]. Available: https://github.com/fchollet/keras

[35] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[36] NVIDIA, "NVIDIA cuDNN: GPU Accelerated Deep Learning," [Online; accessed January 11, 2018]. [Online]. Available: https://developer.nvidia.com/cudnn

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

[38] N. Numao, S. Yoshida, Y. Komai, C. Ishii, M. Kagawa, T. Kijima, M. Yokoyama, J. Ishioka, Y. Matsuoka, F. Koga, K. Saito, H. Masuda, Y. Fujii, S. Kawakami and K. Kihara, "Usefulness of pre-biopsy multiparametric magnetic resonance imaging and clinical variables to reduce initial prostate biopsy in men with suspected clinically localized prostate cancer," *The Journal of Urology*, vol. 190, no. 2, pp. 502 – 508, 2013.

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

[40] "MATLAB 2016b," The MathWorks, Natick, MA, USA.

[41] T. Lin, P. Goyal, R. B. Girshick, K. He and P. Dollár, "Focal loss for dense object detection," *CoRR*, vol. abs/1708.02002, 2017.

[42] Canadian Cancer Society, "The prostate," [Online; accessed January 09, 2018]. [Online]. Available: http://www.cancer.ca/en/cancer-information/cancer-type/prostate/prostate-cancer/the-prostate

[43] J. Joy Lee, I.-C. Thomas, R. Nolley, M. Ferrari, J. Brooks and J. Leppert, "Biologic differences between peripheral and transition zone prostate cancer," vol. 75, 02 2015.

[44] J. P. Hornak, "The Basics of MRI," 1996, [Online; accessed February 20, 2018]. [Online]. Available: https://www.cis.rit.edu/htbooks/mri/

[45] I. L. Pykett, J. H. Newhouse, F. S. Buonanno, T. J. Brady, M. R. Goldman, J. P. Kistler and G. M. Pohost, "Principles of nuclear magnetic resonance imaging." *Radiology*, vol. 143, no. 1, pp. 157–168, 1982, pMID: 7038763.

[46] Wikipedia, the free encyclopedia, "Brain MRI," 2014, [Online; accessed February 23, 2018]. [Online]. Available: https://commons.wikimedia.org/wiki/File:Murphy_2013_brain_MRE_with_wave_image.png

[47] NVIDIA, "GeForce GTX 1070 family," [Online; accessed January 11, 2018]. [Online]. Available: https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1070