Christoph Kaltenböck

Head Detection for Sleep Motion Analysis

Master's Thesis

Graz University of Technology

Institute for Computer Graphics and Vision

Supervisor: Univ.-Prof. Dipl-Ing. Dr.techn. Thomas Pock

Graz, July 2018

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, ____________________ ____________________

Date Signature

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am ____________________ ____________________

Datum Unterschrift

[1]Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Abstract

In this modern world sleep problems have become an important medical problem in the society. A lot of people suffer from sleep irregularities, insomnia and other disease. This has an impact on the physiological and psychological stability of the persons. For investigating the cause of this sleeping problems people have to go to sleep laboratories.

New technology can support the investigation process in the sleep laboratory. With the help of 3D cameras certain body parts can be detected and monitored during the whole night of sleeping.

The goal of this thesis is to develop an algorithm, which is able to detect these body parts for a further processing. It especially concentrates on the mainly visible part on top of the person, the head. Two different algorithms are designed to reach this goal and are able to detect the position and shape of the head of a person lying in a hospital bed.

A Two Stage Approach was designed as first algorithm. There the head can be found with the help of an infrared and a depth image of the individual. The first stage calculates a rough estimation of possible positions of the head. In the second stage the proposed positions are more refined and finally end up in a defined position of the head.

The second algorithm uses Convolutional Neural Networks to train a model, which learns the weights of the various layers with the help of training data. Well-known libraries as Keras and Tensor Flow are used as a base for the implementation. Two different model architectures are used, where the first, Patch Based Network, trains from extracted patches and the second, Fully Convolutional Network, learns a heatmap from the full depth image and its corresponding likelihood map.

Both algorithms have their advantages and disadvantages and perform well on the dataset. The Convolutional Neural Network and especially the Fully Convolutional Network show detection rates of 99% on unseen testing data. In the future the state-of-the-art algorithm, which uses Convolutional Neural Network, should gain more focus in development as the Two Stage Approach. Additional body parts can be added with new annotated areas in the images, as well as parameter tuning and increasing the dataset helps to strengthen the neural network.

Kurzfassung

Schlafprobleme sind in den letzten Jahren zu einer großen medizinischen Herausforderung geworden. Viele Menschen leiden an speziellen Krankheiten wie der Schlafapnoe, Insomnia oder ähnlichen. Um dem entgegen zu wirken, werden die Patienten über Nacht in einem Schlaflabor beobachtet. Miit den gewonnenen Resultaten kann die entsprechende Medikation verschrieben, oder anderwärtig medizinisch nach gebessert werden.

Eine 3D Kamera und das daraus gewonnene Bildmaterial unterstützt den Prozess der Detektion und Lokalisation der Probleme und hilft somit dem Diagnostiker entsprechende Maßnahmen zu setzen. In dieser Arbeit wird eben jenes Bildmaterial als Ausgangspunkt verwendet um eines der wichtigsten Körperteile, den Kopf, korrekt aus einem Tiefenbild, dass direkt in einem Schlaflabor erzeugt wurde, zu erkennen.

Dafür wurden zwei grundsätzliche Algorithmen entwickelt: Der erste, Two Stage Approach, extrahiert verschiedene Merkmale im Tiefen- und Infrarotbild, kombiniert diese und erhält eine Schätzung des Umrisses und der Position des Kopfes. In der ersten Stage wird durch Mustererkennung eine Anzahl von möglichen Positionen, wo sich der Kopf befinden könnte extrahiert. Die zweite Stage bindet sowohl die Werte im Infrarot Bild ein als auch die Eigenschaften der Kontur um aus den möglichen Positionen die korrekte Position zu errechnen.

Im zweiten Algorithmus wird auf Convolutional Neural Networks gesetzt. Anstatt die Parameter vorab zu definieren, werden sie im Laufe des Trainingsprozesses vom Netzwerk gelernt. Zwei Grundarchitekturen, Fully Convolutional Network und Patch Based Network, wurden mit verschiedenen Modellen evaluiert und final wurde eine Detektionsrate von 99% auf das verfügbare Validierungsset erreicht

Acknowledgments

First I want to thank my supervisor Prof. Dr. Thomas Pock for his commitment, his patients due to the long phase of writing this diploma thesis and the tips he gave me during the whole implementation and experimentation phase of the work. As second important person I want to thank my supervisor from the Austrian Institute of Technoloy, Bernhard Kohn, for long discussions about algorithms, input during and after the work at AIT and being patient with my way of working and writing this thesis.

I want to thank my study colleges and especially Jakob Auer for kicking up my ass for bringing this to an end and the bed and workplace, which always was ready to stay a night or two in Graz.

Last, but most important I want to thank my dear girlfriend Daniela for supporting me during the whole long process and never letting me give up.

Thanks to all mentioned above and everyone else in my environment for motivating me again and again to finish this thesis after three years of ups, downs and long breaks in working and writing it.

Christoph

Contents

Abstract v

Kurzfassung vii

Acknowledgments ix

1 Introduction 1
- 1.1 Contribution of this work 1
- 1.2 Goal 1
- 1.3 Organization 3

2 Background 5
- 2.1 Medical Background 5
  - 2.1.1 The laboratory setup 5
  - 2.1.2 Disease detection 5
- 2.2 Technical background 6
  - 2.2.1 Setup 6
  - 2.2.2 Dataset 7
  - 2.2.3 Tools 9

3 Related Work 11

4 Two Stage Approach - A filter and feature extraction algorithm 13
- 4.1 Stage one – Kernel Matching 13
  - 4.1.1 Filtering 13
  - 4.1.2 Maxima Extraction 15
- 4.2 Stage Two – Additional Ratings 16
  - 4.2.1 Infrared Ranking 16
  - 4.2.2 Filter Rating 17
  - 4.2.3 Feature Rating 17
  - 4.2.4 Result – Head Position 20
- 4.3 Result and Discussion 20

5 Deep Learning 23
- 5.1 Introduction to Deep Learning 23
  - 5.1.1 Multilayer Perceptron 24
- 5.2 The Convolutional Neural Network 26
  - 5.2.1 The convolutional layer 27
  - 5.2.2 Activation function 28
  - 5.2.3 Loss Function 30

5.2.4 Optimization and Backpropagation 30
5.2.5 Regularization and additional Functions 32
5.2.6 Regression vs. classification 36
5.3 Models 37
5.3.1 Patch Based Network 37
5.3.2 Fully Convolutional Network 41
5.4 Data Preparation and Augmentation 44
5.4.1 Data selection 44
5.4.2 x Data Preparation 45
5.4.3 Data Augmentation 49
5.4.4 y Data Preparation 49
5.4.5 Data storing 51
5.5 Validation, Training and Testing 52
5.5.1 K - fold validation 52
5.5.2 Training 53
5.5.3 Testing 53
5.6 Results 54
5.6.1 Fully Convolutional Network 54
5.6.2 Patch Based Network 56
5.7 Interpretation 57
5.7.1 Fully Convolutional Network 57
5.7.2 Patch Based Network 59
5.7.3 Summary 63

6 Conclusion and Future Work **65**

Bibliography **69**

List of Figures

1.1 An example of a sleep labratory patient 2

2.1 The technical setup 7

4.1 The two stage algorithm approach 14
4.2 Kernel 26px x 26px with 2D Gaussian with σ = 26 14
4.3 Feature map example 15
4.4 Exampe of a baldhead test person 16
4.5 Exampe of a infrared ranking extraction 17
4.6 Kernel with σ = 26 and σ = 24 18
4.7 Learned Kernel with a lenght of 26 pixel and 32 pixel 18

5.1 Multilayer Perceptron 25
5.2 Plot of a sigmoid activation function 26
5.3 Plot of a hyperbolic tangent activation function [28] 27
5.4 Image Patch Convolution 28
5.5 The rectified linear unit function, ReLu 29
5.6 Underfitting, Overfitting, and good correlation 33
5.7 Dropout in a general neural network 34
5.8 Example of Max Pooling 35
5.9 Patch Based Network - Model P1 38
5.10 Patch Based Network - Model P2 40
5.11 Fully Convolutional Network - Model F1 41
5.12 Fully Convolutional Network - Model F2 43
5.13 Fully Convolutional Network - Model F3 44
5.14 Sample of a depth frame directly from the Kinect camera 45
5.15 Sample of an interpolated original depth frame 46
5.16 Sample of interpolated and downsized depth frame 47
5.17 Sample of interpolated and downsized and padded depth frame for PBN 48
5.18 Sample of interpolated, downsized, padded, and cut depth frame for FCN 48
5.19 Output representation of a sample likelihood 51
5.20 Fully Convolutional Network validation loss progress of all three models 58
5.21 Fully Convolutional Network accuracy of all three models over the predicted likelihood threshold of the head 58
5.22 Positive example of a predicted test image with Model F1 in FCN. 60
5.23 Positive example of a predicted test image with Model F1 in FCN. 60
5.24 Positive example of a predicted test image with Model F1 in FCN. 61
5.25 Positive example of a predicted test image with Model F1 in FCN. 61
5.26 Patch Based Network validation loss progress of the two models 62

5.27 Patch Based Network accuracy of the two models over the predicted likelihood threshold of the head 63

1 Introduction

Recent reports indicate that in the last five years the nights people spent in sleep laboratories increased at four percent each year. During at least one night a polysomnography is written. This is a multi parametric test with at least 12 parameters and contains 22 wire attachments to the body of the patient. With the help of this test doctors can identify and define different kinds of disease. On the one hand the whole procedure is necessary to make a diagnosis and settle a treatment plan, but on the other hand also challenging and uncomfortable for the patients. Figure 1.1 shows an example of a person, who lies in a sleep laboratory.

People who come to a sleep laboratory do already have sleep issues before they come to the hospital. The combination of a different environment and the wires, mask and other instruments, makes sleeping in the laboratory even worse. This is not only bad for the patient, also the result of the polysomnography can be falsified due to the inconvenience of the sleep.

1.1 Contribution of this work

This work shows two different algorithms for the detection of the shape and position of the head. While the rest of the human body is partly covered by the blanket the head of a sleeping person is the only part of the body which is visible in nearly all sleeping positions. An accurate position and shape detection can then lead to a better torso estimation and helps for following calculations.

Following algorithms are used:

- The first is a Two Stage Approach. A kernel patch with the 2 1/2 D model of a head is matched over the whole image. The best matching part in the image therefore defines the location of the head. Further processing also brings information about the shape.
- The second method uses supervised machine learning: A Convolutional Neural Network, short CNN, was trained to find the position and shape of the head in the image.

1.2 Goal

Different algorithms should detect equivalent data as the Polysomnography via data from a 3D camera. Getting rid or reducing the amount of wire attachments leads to a better and more realistic sleep of the patient and improves the final results. Moreover, disease detection is done quicker

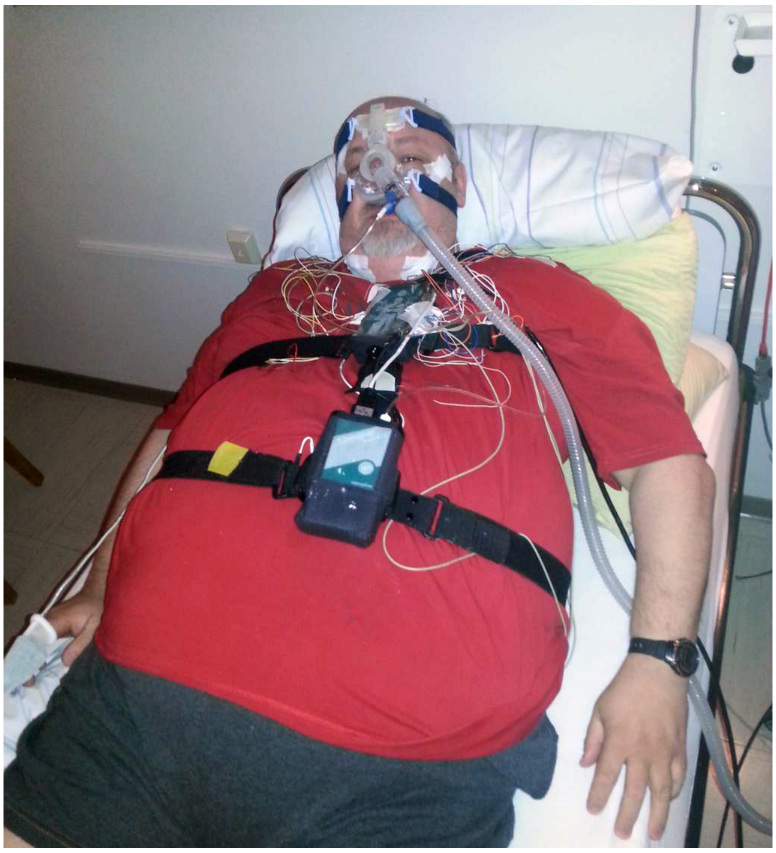

Figure 1.1: A patient who is prepared for the polysomnography. The CPAP (Continuous positive airway pressure) mask around the mouth of the patient is necessary for further treatment [55]

and more accurate. People who normally have to stay a night in a sleep lab can then bring the camera and surrounding setup to their home and install the equipment at their own bed in their personal environment. The sleep would neither be disrupted by wires, masks and the rest of the setup, nor by and unfamiliar environment for the patient. This produces optimal conditions for the detection.

The goal of the whole project is to make sleep laboratories more comfortable for the patient on the one hand and the results more reliable on the other hand. This works intention is to detect the position and shape of the patients head to further detect other body parts and refine the positioning of measurable areas alongside the patient.

1.3 Organization

This Introduction is followed by a chapter about background information. Medical diseases which can be detected with the help of the provided work are presented as well as the experiment setup, used dataset and tools. The next chapter shows work which is related and was done by other researchers. The fourth and fifth chapters deal with the two methods Two Stage Approach and Convolutional Neural Network. They contain detailed information about the calculation and implementation together with results and interpretation. In a Conclusion and Future Work chapter the results of both methods are compared and analyzed. Improvements for the algorithms and further projects withn the range of this work are also presented in this chapter and end up the diploma thesis.

2 Background

The first Section Medical Background gives an overview on the medical part of the work. It describes the aim of the study and how sleep laboratories currently work. In the second Section the technical part is presented.

2.1 Medical Background

In the year 2016, 273 hospitals were listed in Austria [25]. Among them are thirteen hospitals in which one or more sleep laboratories are operated. All together Austria has 34 laboratories [68] . For this research the three hospitals AKH Wien, Barmherzige Brüder Wien and AKH Linz where picked as test environment.

2.1.1 The laboratory setup

A sleep laboratory itself consists of two different rooms. The first one is the patient's – room, which is provided with a bed and an extra bathroom. The monitoring and recording equipment is located in the second room, which is most likely located nearby.

The art of sleep study is called Polysomnography, short PSG. It is a multi parametric test to diagnose sleep disorders of the individual. The duration of the recording depends on the patient and circumstances and varies between eight and twelve hours. During this time, most likely in the night, the sleep and wake cycles of the individual gets recorded. The recorded data itself is called Polysomnogram and consists of different channels, which can also vary due to the hospital/ sleep laboratory and/or disease to diagnose. Brain activity, eye movements, heart rate, blood pressure, breathing activity and effort and oxygenation status are the most common. [38]

2.1.2 Disease detection

The PSG is designed to detect several kinds of disorders, including Narcolepsy[51], Idiopathic Hypersomnia[62], Periodic Limb Movement Disorder (PLMD) [57], REM Behavior Diisorder [72], Parasomnias [42], and Sleep Apnea [67].

Sleep Apnea Sleep Apnea is the most important disease for the research covered by this work. There are two different kinds of sleep apnea. The first is called Obstructive Sleep Apnea and is caused by a repeated obstruction of the upper airways during sleep, thus leading to a reduced or complete stop of the airflow. Central Sleep Apnea on the other hand arises from missing nerval signals and is a malfunction of the brain.

Continuous Positive Air Pressure (CPAP) is used to treat Sleep Apnea. It is a form of positive airway pressure ventilation. Mild air pressure is constantly held to the patients mouth to keep the airways open. The CPAP mask is often visible on the test data in this work, which causes disruption of the image. [48]

Restless Leg Syndrome Restless Leg Syndrome, short RLS, is a disorder that causes a strong urge to move the legs. This often goes along with strange and unpleasant feeling in the legs. People who have RLS describe it with the words: creeping, crawling, pulling, itching, tingling, burning, aching, or electric shocks. [60] RLS cannot be detected via PSG, but can be detected via image processing and is therefore important for this work.

2.2 Technical background

2.2.1 Setup

Figure 2.1 shows the experimental setup for collecting data. The Kinect for XBox one, short Kinect, camera is positioned above the patient's bed via mounting it on a rack. Furthermore it is connected to a PC where data gets recorded. In addition to the PSG recording, the doctor who monitors it also starts the recording of the image data from the Kinect. Later on they get synchronized, for better research with the help of the Polysomnogram data.

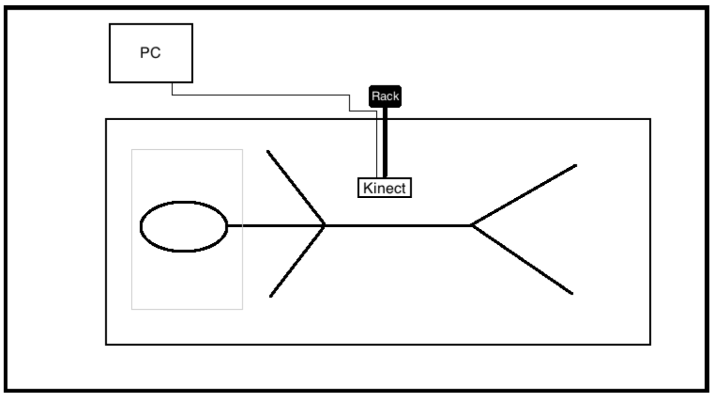

Figure 2.1: The patient lies on the mattress of the bed. About one and a half meter over the top edge of the mattress the Kinect is positioned. It is mounted on a rack, which stands sideways the bed. The angle of view of the camera is at least the full bed plus an additional uninteresting surrounding area. The camera is connected to a computer where the user triggers the record process

2.2.2 Dataset

Recording The available data comes from two different environments. The first dataset is directly extracted from real life scenarios. Patients from three hospitals in Austria (Barmherzige Brüder Wien, Allgemeines Krankenhaus Wien and Algemeines Krankenhaus Linz) have given their approval to record video in addition to the channels of the PSG. Fiftyfive persons got recorded during one night over approximately eight hours. Infrared and depth image were shot at 30 frames per second. The individuals have various laying positions and different coverage of the blanket. Some patients had to wear a CPAP – mask, others had a bar over their head, which covers the complete head sometimes.

The second dataset was recorded in a controlled environment. A test laboratory has been created with a mattress, the Kinect camera and the recording pc. Seven test persons were available for recording. These persons were told to get into different sleep positions. The personal habits of them contribute to the diversity of the dataset. They had different physical characteristics: Both sexes, skin colour diverges from white to dark brown, hair colour from blond to black and also one person with a bald head is in the test set.

A storyboard was written for recording the sessions. The test person had to get into the following scenarios, which were all combined with each other.

Blanket:

- without blanket
- with blanket, arms outside
- with blanket, only head outside

Position body:

- beach chair position
- on the left side
- on the right side
- abdominal position

Position feet (beach chair position only):

- fold to the right side
- fold to the left side
- fold straight to the body
- stretched

Head position:

- left
- straight
- right
- on the pillow / not on the pillow

Data is stored via the Hierarchical Data Format, short HDF [84]. Infrared and depth image are saved in separated data fields in a three dimensional stack (x and y axes and time).

Annotation To simplify the annotation process, the form of the head was assumed to be an ellipse. With a script based, self-written, helper application called Person Label Tool [32] the images got labelled by hand.

Images from the first dataset (with the real life data) got labelled randomized. That means the Person Label Tool shows the annotator a randomly selected image of one patient and he/she labels the head via positioning and changing the shape of the ellipse.

For each person in the second dataset, 200 images where selected in equal distances. This covers at least one snapshot of a position of the test person.

All annotation data are stored as JSON[27] files. There is one file per person. The number of the frame identifies the image on the timeline of the night in the sleep laboratory / time in the test laboratory and is used as root element in the JSON tree. One step below the tag `head` marks the root element of the hierarchical tree. Further annotations could be added via different elements (e.g.: `body` or `right hand`). As actual data the ellipse is described via the position in x and y coordinates, the size of the ellipse via a (semi-major axis) and b (semi-minor axis) values [43] and the rotation in degrees. The angle is measured at the bottom left corner of the surrounding box from the horizontal baseline to the beginning of the box. Listing 2.1 contains an example for two datapoints in an annotation file.

Listing 2.1: JSON - File Snipped of one patient

```
{"149461":
     {"head": [[87, 125], [48, 35], -544.23]},
 "420067":
     {"head": [[96, 118], [48, 35], -549.04]}
}
```

2.2.3 Tools

The programming language Python was used for developing the algorithms. Amongst others the free libraries Numpy[53] and SciPy[31] were used for mathematical calculations. For the visualization and graphical user interface PyQtGraph[58] and Matplotlib[26] was utilized. The framework Keras got used for implementing the Convolutional Neural Network part and HDF [84] and Pandas[47] for file and data handling. All sources got implemented as scripts and use file interfaces for sharing among themselves.

As 2 1/2 D camera for recording the scene in the laboratory the Microsoft Kinect v2 or Microsoft for XBox one was used. A script reads the infrared and depth frames from the Microsoft SDK[33] and writes them out in an HDF file.

3 Related Work

Clabian et al use a sparse 3D data set (19x19 data points) to construct 3D surfaces over the image plain and detect ellipsoid shapes out of it which therefore can be determined as head shape. The detection rate goes up to 90% of the tested images. [11]

Xia et al use a two stage approach on the depth frame of the kinect to detect the head: In the first stage edges in the 2D image were detected, in the second stage a 3D shape detector was used. Thus it was possible to segment the human figure out of the background.[85]

Wang and Hunter did research on body part detection in a medical environment. They present a novel approach to detect and track the upper body parts of a covered body in a bed. In the former paper the algorithm is based on 2D images. Besides a noise proof preprocessing, a head tracker and special torso finder they introduced an own hierachical-boosting head detector. This head detector contains three hierachical layers based on preprocessed edge images and works with a pattern based machine learning algorithm. In the later paper they show improved algorithms. Amongst others they use different filters for the edge detection in preprocessing the images and a case structure for the model to train. [79][80]

Chen et al. used only depth information from a Kinect camera to train a head descriptor for determining each pixel as head or not head pixel. Head pixels were clustered to reduce computation time and 90% accuracy was achieved for their dataset which contains different body postures (also partly occluded head parts). [9].

The main application field in Deep Learning are Convolutional Neural Networks. Szegedy et al [74] show a Convolutional Neural Network approach for object detection and localisation on the PASCAL VOC [21]. They get good results compared to similar and state - of - the- art algorithms.Long et al [39] developed a network, which creates a heatmap of where different objects can be located in a RGB image. For this application they used a Fully Convolutional Network and the power of the whole image. On the PASCAL VOC 2011 and 2012 they got an improvement of 20% with their algorithm. Diego et al [17] also used the power of a Fully Convolutional Network to detect head of persons in a depth image via probability map. They gained 98.2% Accuracy on the Pandora [3] dataset.

4 Two Stage Approach - A filter and feature extraction algorithm

The algorithm is based on a two-stage approach. The first stage selects a small number of possible head positions. These positions are particularly handled in the second stage. After the second stage there is a probability rating for each selected position from the first stage. The highest value determines the highest probability to be the correct head position within the whole image. Figure 4.1 shows an abstract illustration of the Stage Model. The following sections show how these stages are implemented and how the selection process works.

4.1 Stage one – Kernel Matching

Filter operations often are used in image processing. They are used for smoothing and blurring images, edge detection, noise reduction, super- resolution and others. In this work a filter was constructed to detect head shapes in the depth image from the Kinect camera.

4.1.1 Filtering

Viewing the patient from the top point of the bed (see Technical Setup in Section2.2.1) let one imagine how a head filter should look like. On the top of the elliptic head shape the distance from the head to the camera is lowest. Away from this point the distance rises to the point, where the head touches the mattress or the pillow. Since patients do not lie still in a horizontal position, the ellipse always has a rotation in a direction and a different size. To keep the filtering easy and independent for all different kinds of heads and sleep positions, a quadratic form is used. Figure 4.2 shows the 26x26 pixel filter. It was created using a Gaussian Curve with $\sigma = 26$.

The kernel is windowed over the entire depth frame. That means the algorithm begins in the upper left corner of the image and iterates over each pixel. The distance between this pixel and the center of the kernel gets calculated and set as reference value. The distance of the surrounding values get also measured and subtracted by the reference value. The sum over all distances should get minimal, then the best fitting patch was found and should in theory be the position of the true head. Equation 4.2 shows how this works. ref_{xy} represents the mentioned reference value. I is the depth frame, K the kernel and $f(x,y)$ is the resulting feature map value on the position x and y. Undefined values from the camera (dead pixels) get ignored in the calculation. For optimizing the

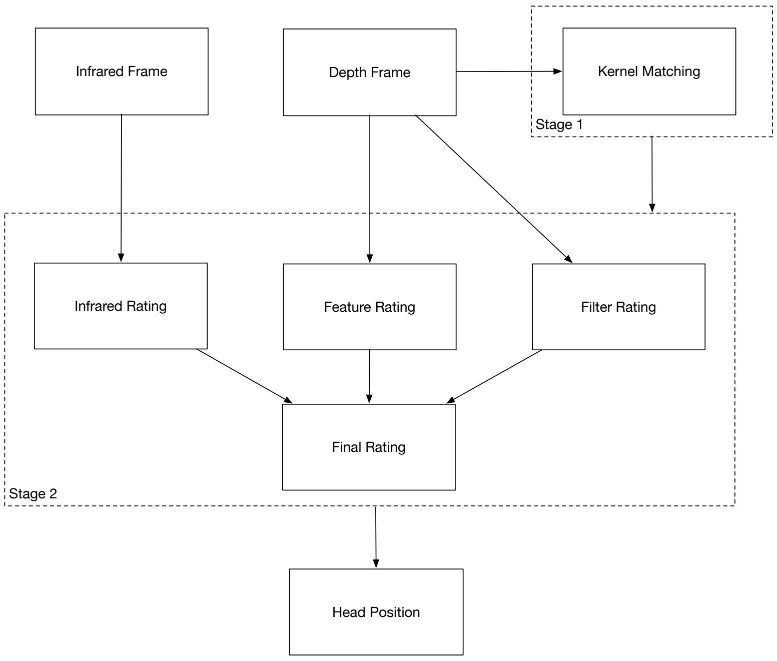

Figure 4.1: The two stage algorithm approach

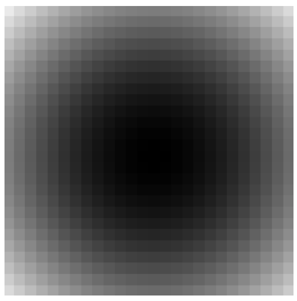

Figure 4.2: Kernel 26px x 26px with 2D Gaussian with σ = 26

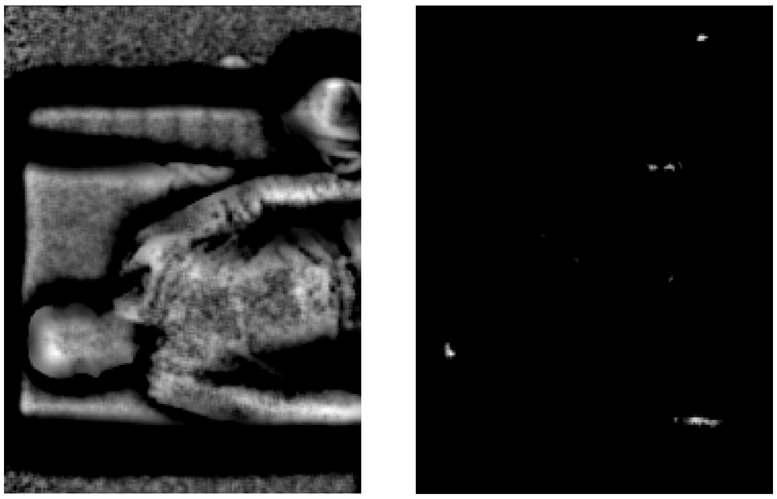

Figure 4.3: An example of the feature map which accrues after filtering. On the left side one can see the result: The shape of the body can be recognized. The right side shows the maxima of the feature map.

process only the left half of the image is taken. It is assumed that the head of the individual is always located in this area of the frame.

$$ref_{xy} = |I(x,y) - K(x_0,y_0)| \tag{4.1}$$

$$f(x,y) = \frac{\sum_{k_x=-x_0}^{x_0} \sum_{k_y=-y_0}^{y_0} \left|I(x+k_x, y+k_y) - K(k_x,k_y) - ref_{xy})\right|}{4x_0y_0} \tag{4.2}$$

4.1.2 Maxima Extraction

As a result of the filtering a 2D feature map accrues. An example is depicted in figure 4.3. In the left image the person is recognisable - The white/grey parts mark the better matching pixels, the black parts mark the pixels that do not match. The right image shows a cut version and the maxima regions, which are interesting for further investigations.

With the SciPy filter method `maximum_filter` and `minimum_filter` one gets all maxima and minima with a predefined count of neighbourhood pixels. If the difference between a maximum and a minimum is bigger than a certain threshold the maximum is utilized for further processing.

Testing the extraction of the algorithm at the first dataset has shown, that at a neighbourhood of 40 pixels (for the case of a filter size of 26px times 26px) the maximum, which is located in the area of the head is within the five highest local maxima of the image. Therefore extracting five maxima for further processing in the next stage is enough.

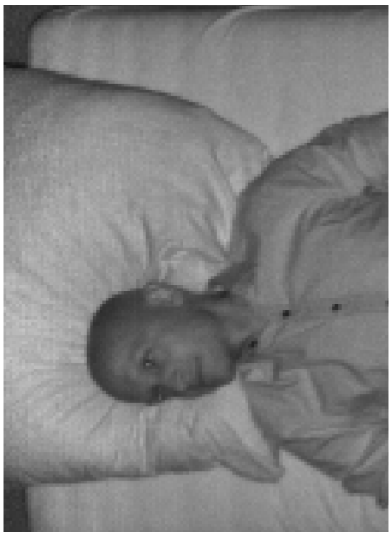

Figure 4.4: An example of a baldhead test person. The parts of the baldhead are still darker than the rest of the body

4.2 Stage Two – Additional Ratings

Stage one resulted in five possible head positions. These positions are now considered more precisely. Three different algorithms deliver a separated ranking for the correct head position. The merge of these results provides the end result and position of the head.

4.2.1 Infrared Ranking

One third of the final rating uses the infrared image, which also is provided from the Kinect camera. On the test images we saw, that the head of the test person always has darker areas than parts of the clothing, bed, pillow or blanket. Figure 4.4 shows, that this also works for people with baldhead. The awareness of this phenomenon is, that a patch around the maxima positions is used to rank them from lightest to darkest, where the darkest area is the most probable head position. Figure 4.5 presents the values of the extracted infrared patches. The smaller the values are, the darker and higher ranked they are. Equation 4.3 shows the mathematical calculation where I_{range} is the range around the maximum, which is considered (in this case 13 pixel were set), I is the infrared image and $R(x_m, y_m)$ is the calculated value for the given maximum.

$$R(x_m, y_m) = \sum_{x=x_m-I_{range}}^{x_m+I_{range}} \sum_{y=y_m-I_{range}}^{x_y+I_{range}} I(x,y) \tag{4.3}$$

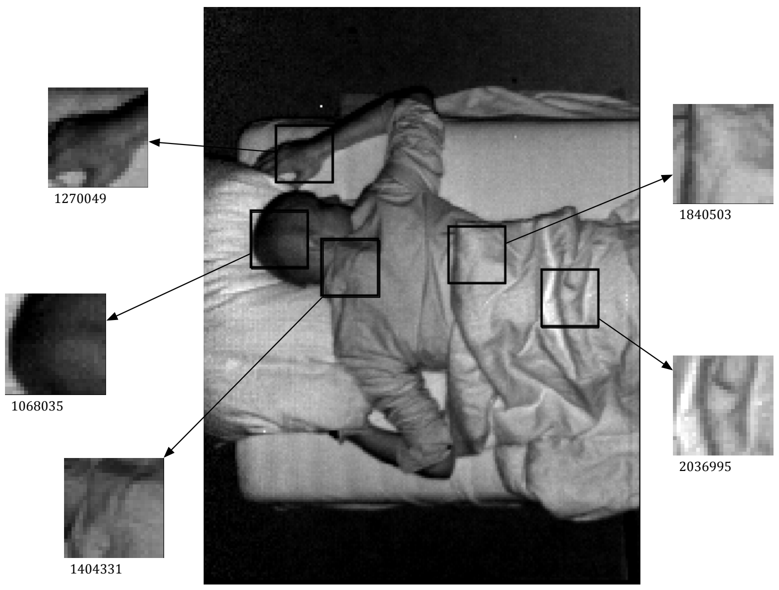

Figure 4.5: An example of an infrared image with a person, where the patches of the five most interesting maxima of stage one got extracted and rated. The smaller the number the darker the area and has therefore a higher probability to be the real head position

4.2.2 Filter Rating

Windowing over the whole image (or even over the half image) is expensive. With the knowledge of stage one, selected points can further be filtered with different kernels. The outcome of the filtering presents the second of three ratings for the final head detection.

For each of the extracted maxima, every filter is applied. The procedure is the same as described in stage one with the one kernel (See Section 4.1 but this time only for the special pixel in the image, which marks the maxima from stage one. The resulting values of all filter operations get summed up for every maximum and the resulting value is the rating for the maxima.

Figure 4.6 and Figure 4.7 show the different kernels, which got used for this algorithm.

4.2.3 Feature Rating

The third part of the second stage takes the contours of the surrounding area of the maxima into account. In the Seminarproject [32] an approach was shown, that uses the properties of extracted contours in a sliced model of the depth frame to identify if the contour looks like a head form or not. The result there did not really work as expected but the idea was taken as an additional feature to add another variation to the rating.

The stage one algorithm results in a map where each pixel determines how good the position fits

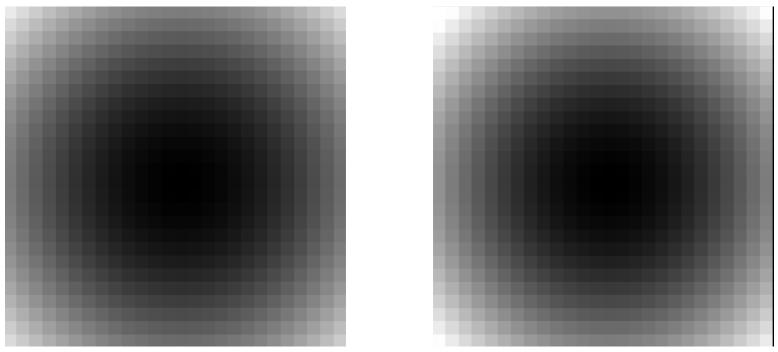

Figure 4.6: The left image shows the kernel with σ = 26, which was already used for the maxima extraction. On the right image the kernel with a different σ is shown. The σ was changed to 24 for changing the distribution of the gaussian and therefore the form of the head to fit

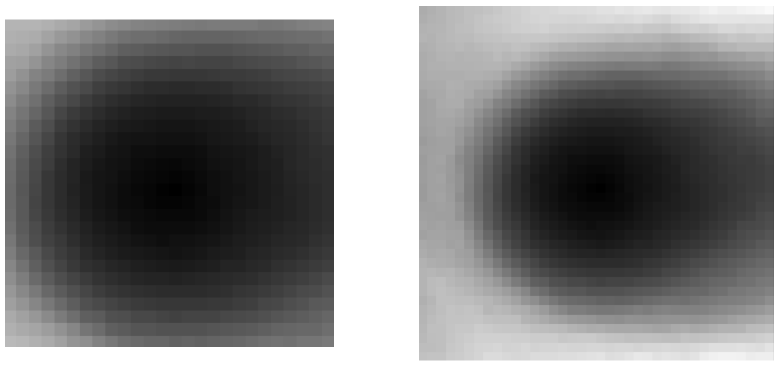

Figure 4.7: These two kernels are calculated with available data. All annotated frames from the real hospital dataset were taken into account for producing these kernels. The kernel size defines the values in the image, which were taken for the calculation. The 26pixel times 26pixel kernel (left image) takes 13pixel in the minus x and 13 pixel in the plus x direction as well as 13 pixel in the minus y and 13 pixel in the plus y direction. The median over all images for every pixel is built and taken for the kernel. That means the value on K(0,0) is defined by the values of median(I1(x0-13,y0-13), I2(x0-13, y0-13),. . .) For the 32pixel kernel (right image) the range is from -16pixel to +16pixel. This results to a good representation of the area around the head positions and therefore to a good estimation of a kernel.

to the kernel and therefore it votes for a correct head position. This head positions define only the peak of the head. As the map is scaled from zero to one a simple threshold helps to separate the surrounding pixels of the maxima from the non head pixels and creates a contour which represents the head shape. In this algorithm a threshold value of `HEAD_RANGE=0.01` was used.

As described in the Seminarproject, different features are extracted from each of the contours. Every feature has a value, which fit to a reference head shape. A gaussian curve determines how far away the measured value is from the reference value. Equation 4.4 shows how the final feature is built.

$$p_w = e^{\frac{-(fw - fwref)^2}{2\sigma^2}} \tag{4.4}$$

Table 4.1 shows which features are used, their reference value and the σ for calculating Equation 4.4.

Feature	f_{wref}	σ
Aspect Ratio	0.6	0.4
Extent	0.7	0.4
Solidity	0.93	0.05
Equivalent Diameter	22	12

Table 4.1: Feature standard valuesf_{wref} and σ Value

The definition of different σ values helps to weigh better working features more than less good features. The following Section gives an overview for the used features [1]

Aspect Ratio The Aspect Ratio is calculated from the ration of width and height of the shapes contour.

Extent Ratio of pixels in the region to pixels in the total bounding box. Computed as `area / (rows * cols)`

Solidity Ratio of pixels in the region to pixels of the convex hull image.

Equivalent Diameter The diameter of a circle with the same area as the region.

The location of the contour is added as an additional feature. It gets a value from zero to one according to how far away the contour is from an expected point of the head in the bed.

[1]http://scikit-image.org/docs/dev/api/skimage.measure.html

The final Contour feature is calculated via averaging over all five features. Equation 4.5 shows the final calculation.

$$rating = \frac{(p\_aspectRatio + p\_extent + p\_solidity + p\_equi\_diameter + location)}{5} \tag{4.5}$$

4.2.4 Result – Head Position

All three sub-ratings get normed from zero to one and summed up for a final rating. The highest ranked maxima determine the found head position. The contour, which was extracted, can be used as the shape of the head.

4.3 Result and Discussion

The algorithm was developed by constantly observing the head detection rate at 357 annotated frames from different real life patients (First dataset). The performance of a feature on this set of data decided if the feature came into the algorithm and which parameters were set for the feature. The following paragraphs show the decision that were made by reference to their results:

Kernel Matching Kernel The goal was, that as many as possible head positions can be detected via windowing one kernel over the depth frame. Two criteria were important: The number of frames, where the head got detected with the first extracted maxima should be maximal and the head of all frames should be detected in a minimal range of extracted maxima. For example a Kernel which detects nearly all head positions with the first maxima but does not detect the head in one frame in any of the maxima is worse than a Kernel, which detects every head position within the maxima and fewer head positions in the biggest maxima.

All four Kernels got tested and the result is listed in Table 4.2. The second column shows the accuracy of the heads, which were detected in with the first maxima. The third column represents the count of frames, where the head is not located in any extracted maxima.

Kernel	Acc	not detected
26px x 26px, σ = 26	82.3%	0
26px x 26px, σ = 24	79.3%	0
26px x 26px, Median over all frames	79.6%	1
38px x 38, Median over all frames	88.0%	4

Table 4.2: Results of different Kernels

The 26px x 26px kernel with σ = 26 was chosen for the primary extraction because the head positions of all 357 frames were detected inside the five best maxima and it performed better than the kernel with $\sigma = 24$. The correct detected frames of one kernel differ from the correct detected frames of the other kernels this lead to the idea of combining them to an additional feature.

The combination of all four Kernels results in a detection rate of 86%, which is worse than the 88% of the 38px x 38px Median filter but guarantees that the head position is contained inside the five best extracted maxima.

Infrared Rating The infrared feature applied to the 357 frames results in a detection rate of 81,5% on the pre filtered maxima positions (the biggest five maxima with the 26px x 26px, $\sigma = 26$ kernel). In combination with the additional feature selection a rate of 95.8% can be achieved.

Feature Rating With adding the feature rating to the Infrared Rating and Kernel Matching Rating a final detection rate of 98.9% could be achieved. That means at four of the 357 images the head was not correctly detected by the Two Stage Algorithm.

Testing the Two Stage Algorithm with the dataset of the test persons gives the results which are listed in Table 4.3.

Patient	Detection Rate
CK280116	95.7%
AP280116	87.6%
BK280116	75%
CA280116	56.9%
MC280116	92.6%
ML280116	86.2%
SS280116	90.2%

Table 4.3: Accuracy for different test persons

In 1181 images out of 1416 the head position was correctly found that is an accuracy of 83.4%.

Conclusion In the beginning of the work it was assumed that the algorithm is invariant for different environments and people. This could not be proved as true statement. Other than that the detection rate on the dataset which was used to define parameters and tweak features was good but for new data the results vary a lot depending on the person and his or her habits and physical appearance.

5 Deep Learning

The last algorithm gave good results with using simple feature - based functions. That means one image is used as an input and various features are extracted and rated for getting final results. The features themselves are based on knowledge from observing the test data. For example: A filter was designed with the assumption, that the best fitting form of a head is an ellipse and the further assumption that when the rotation of the head is taken into account this leads to the form of a circle. Instead of defining this features on testing the different test data and adjusting the parameters step by step this could be done automatically. A Machine Learning algorithm would bring this to a higher level and adjust the different features until the result on the training data is optimal. In this section a step further is taken directly into the world of Machine Learning. For solving the task of extracting the position and shape of the head of a human body a Convolutional Neural Network, short CNN, is taken as second algorithm.

Everything in Machine Learning is hierarchically organised: The root element of everything in this area is called Artificial Intelligence, short AI, a sub theme of it is Machine Learning. In the wide area of Machine Learning there are supervised and unsupervised learning [18], as well as reinforcement learning [73]. Supervised learning is further categorized into Shallow Learning and Deep Learning [64]. Where the first basically represents a piping of input data through a feature and getting output data (Example: Linear Regression [65]) and the second pipes input data to a stack of features and gets output data. Deep Learning is used for more complex and generalized topics, Shallow Learning for simpler and straight forward problems. The following Paragraphs concentrates on the topic of Deep Learning and its sub derivatives and describes the way from a simple Neuron, over the Multilayer Perceptron to the Convolutional Neural Network, which is state-of-the art in all image processing tasks.

5.1 Introduction to Deep Learning

Back in the year 1943 two scientists, Warren S. McCulloch and Walter Pitts [46], released a paper where they first described a mathematical representation of neurons in the brain and their funcionality. They assume that neurons are grouped into nets and each neuron, which fires at a certain time depends on the surrounding neurons and leads them to also fire or not. With their assumptions and mathematical descriptions they had born the Artificial Neural Network. A neuron in the sense of McCulloch and Pitts works like one unit in a system, Equation 5.1 shows the mathematical representation of a neuron [30], where f is the activation function for the neuron to fire, represented via a basic step - function [82]. w_j is the weight of the synapses, or connections to previous layers and u is the threshold for firing.

$$y = f\left(\sum_{j=1}^{n} w_j x_j - u\right) \tag{5.1}$$

The grouping of neurons represents different kinds of Neural Networks. It can be distinguished between Feed Forward Networks, where the direction is straight forward from the input to the output layer and Recurrent Network, where loops can appear because of feedback connections. [30]. With the help of input data the connection weights of every neuron can be learned. Two different ways are possible: Unsupervised and supervised learning [18], where the former uses input data and learns the structure and represents it in the network and the later also takes known output data into account for training the network to learn a connection between the input data to get the correct output data. The grandfather of most of state-of-the-art Feed Forward Networks is the Multilayer Perceptron.

5.1.1 Multilayer Perceptron

Back in 1961 Rosenblatt invented the perceptron [61], which is built up on a neuron from the system of McCulloch and Pitts. It is a type of linear classifier, that maps its input $\boldsymbol{x}$ to an output $f(\boldsymbol{x})$, where $\boldsymbol{x}$ is a vector and $f(\boldsymbol{x})$ a single binary value which defines if the input values accompany to a class (one) or not (zero). Equation 5.2 shows how the perceptron works. The vector $\boldsymbol{w}$ represents the weights of the perceptron, these are initialised randomly in the beginning and adjusted during the training phase of the perceptron. b is a bias which is independent from the input data and shifts the output of the perceptron.

$$f(x) = \begin{cases} 1, & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b > 0. \\ 0, & \text{otherwise.} \end{cases} \tag{5.2}$$

A number of perceptrons is then grouped into layers. Each perceptron in the layer has a connection to each perceptron in the next layer (fully connected). This is called a Multilayer Perceptron, short MLP. The simplest MLP has an input layer, one hidden layer and an output layer (Figure 5.1 shows a simple MLP).

Activation Different to the stand alone perceptron of Rosenblatt, each neuron of the MLP has a linear activation function, which results in a number instead of a binary value. There are a lot of different activation functions and they play an important role in the performance on neural networks. Isa et al [29] discussed the impacts of different activation functions on a classification MLPs. Sigmoid (see Equation 5.3 and Figure 5.2) is the initially used activation function for MLPs in the beginning. More research on that topic shows that other functions work better and Isa et al. showed that hyperbolic tangent (see Equation 5.4 and Figure 5.3) results in the best results for their classification problem. Later in this work the relevant ones for the network, which is used here are discussed in a separate chapter (see Subsection 5.2.2)

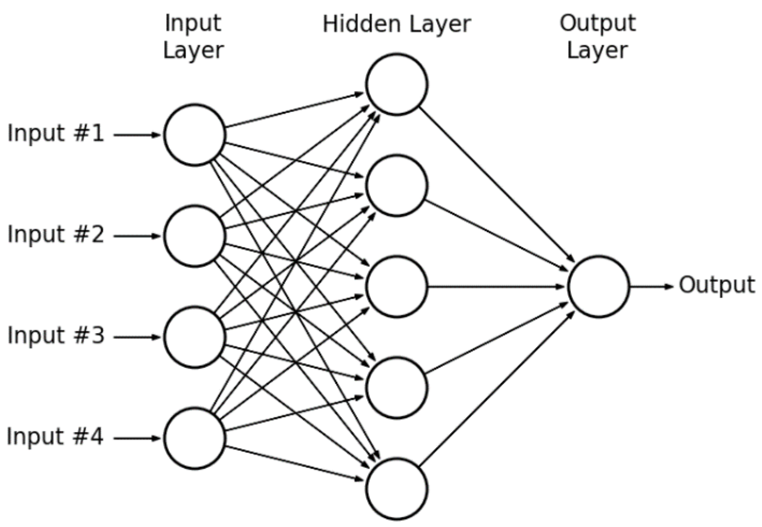

Figure 5.1: The simplest representation of a Multilayer Perceptron with one input layer, one hidden layer and one output layer. Each perceptron from one layer connects to each perceptron in the next layer [49]

Backpropagation The process of feed forwarding data means, that all inputs go through the whole network, beginning at the input layer, going through all hidden layers and ending in the output layer. After passing all layers in the network the output of the output layer (in the simple example: the output perceptron) has one defined value. In the current state of the neural network this would be the result driven by the given input. As the MLP is a supervised machine learning technique, the ground truth output of each passed input is known upfront. The difference between the predicted output value and the known output value give a basic idea of the current error in the network. For optimizing the network this error should be as small as possible. This can be reached via Backpropagation.

Werbos first introduced the term Backpropagation on Neural Networks back in the year 1974[83]. The MLP Backpropagation is done via a generalized version of a least mean squares algorithm in the perceptrons. The weight update has to be done in every perceptron in the neural network. The error for every neuron can be calculated with the following equation. Where $e_j(n)$ is the error in the output node j in the nth datapoint. d is the ground truth value and y the value which the output perceptron delivers.

$$e_j(n) = d_j(n) - y_j(n) \tag{5.5}$$

$$\mathcal{E}(n) = \frac{1}{2}\sum_j e_j^2(n). \tag{5.6}$$

The change in each weight, $\Delta w_{ji}(n)$, is calculated via gradient descent. The Following equation shows the principal calculation, where η represents the learning rate which determines how big the step should be to update the weights and makes sure that the weights converge and do not

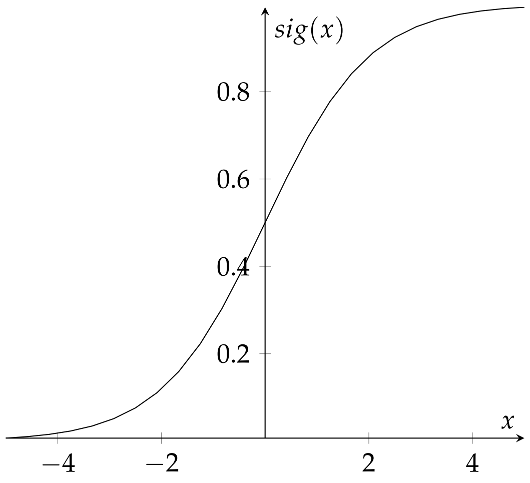

Figure 5.2: Plot of a sigmoid activation function

$$sig(x) = \frac{1}{1+e^{-x}} \tag{5.3}$$

oscillate. y_i is the output of the previous neuron and v_j is the local field which varies.

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial v_j(n)} y_i(n) \tag{5.7}$$

Summary MLP was state-of-the art back in the 1980s and then has not been used any more. As mentioned above it is the grandfather of current Deep Learning networks. Support Vector Machines[13] overcame MLPs back in these days. But as calculation power grew to a certain point and the amount of available data increased as well, the networks of Mc Culloch and Pitts got interesting again: Hinton [24] first introduced the term Deep Learning. The structure of the MLP is the base for every mordern neural network. There is an input layer and an output layer as well as several hidden layers. Each layer delivers the output of itself to the next layer via an activation function. In the end the output layer results in certain values. These predicted values and the known learning output values are combined with a Loss - function to generate the current error of the network. In the backward path, the weights in the neurons are updated via Backpropagation. A certain learning rate in combination with an optimizer (gradient descent based) gives direction and size of the weight update in every neuron. Adding more hidden layers makes the network deeper, this is called Deep Learning.

5.2 The Convolutional Neural Network

Every Deep Learning mechanism, which has to do with images is actually solved with a Convolutional Neural Network. In this Section the details about CNNs are presented. It begins with an explanation of how the convolutional neuron works and how it differs from a neuron in a MLP. Then State-of-the-art activation functions and optimizers for the Backpropagation are presented

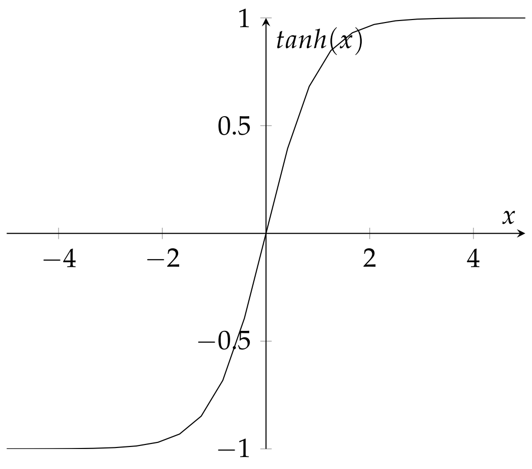

Figure 5.3: Plot of a hyperbolic tangent activation function [28]

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^- x} \qquad (5.4)$$

and shown why and which of them are used along with this thesis. After that regularization and optimization functions are demonstrated. In the end a look at two different training variations, classification and regression, is taken.

5.2.1 The convolutional layer

Filtering is a main part in image processing. Algorithms like the Canny Edge - Detector [6], Hough Transformation [20] , Gaussian Filter[45] or others are widely used in combination, variations and different forms for extracting features out of an image. Fundamentally the method is always the same: A filter with a specified size is windowed over the whole image. A convolution of the filter patch with the underlying image patch gives the value of the certain pixel. Figure5.4 illustrates the convolution on an input image. Equation 5.8 describes the convolution procedure for every pixel (x and y) in the image I, where a and b are the middle points of the kernel (normally it is a quadratic kernel, therefore a would be the same as b), n and m are the edge lengths of the kernel (normally it is a quadratic kernel, therefore n would be the same as m). The convolution reduces the dimensionality of the image (See Paragraph 5.2.5).

$$(I * k)(x,y) = \sum_{i=1}^{n} \sum_{j=1}^{m} I(x - j + a, y - j + b)k(i,j) \qquad (5.8)$$

One neuron in the CNN consists of a convolution from the input with the filter. Different to other filtering techniques in image processing this filter is not fixed - it is learned during the training phase of the CNN. The input layer of the CNN gets the image to train. Each neuron in the layer convolves the image with a filter and pass (after activation) the result to the next convolutional layer. This goes through the whole network and ends in the output of the output layer where the

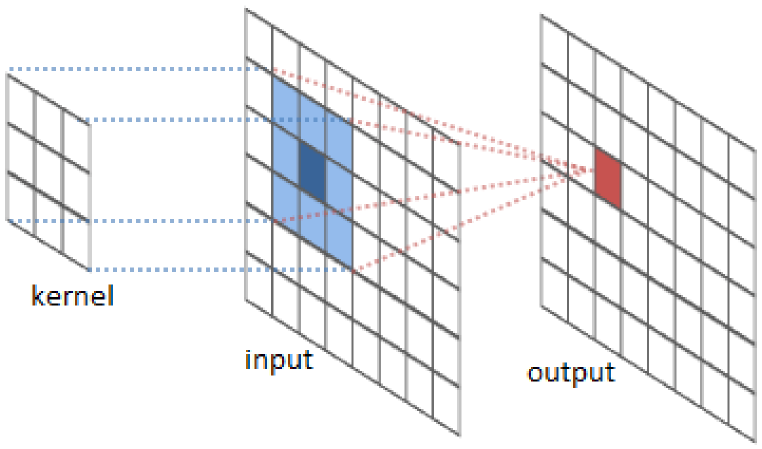

Figure 5.4: A convolution of an input image with a filter patch and the place of the resulting value of the convolution [77]

error is calculated via a certain loss function. Via Backpropagation not the weights, but the filters of the neurons get updated with a defined optimization algorithm.

Every network in this thesis begins with a convolutional layer and then adds additional layers to the network (also more convolutional layers). In Keras this is abstracted as listed in Listing 5.1, where the first parameter defines the number of filters to learn (the neurons in the layer), `kernel_size` defines the size of the filter kernel. The first layer in the network gets the info of the input shape (`input_shape`) of the patch which gets trained. `padding` is for filling the pixels which get lost during the convolution and `strides` define the number of pixels to jump over when convolving. More in Subsection 5.2.5

Listing 5.1: Convolutional layer ReLu

```
model.add(Conv2D(16, kernel_size = (7, 7), padding='same',
    input_shape=(1,img_rows, img_cols)))
```

5.2.2 Activation function

Over the years the activation function got changed a lot. In the basic model of McCulloch and Pitts [46] a single step function determined if (one) or if not (zero) a neuron fires. Investigating more into that, for the MLP a Sigmoid and Hyperbolic Tangent (see Subsection 5.1.1) was defined, where the neuron not only delivers zeros and ones but also everything in between. This also has the advantage, that it is non linear and therefore can be used for the stacking of different layers in the CNN. Both Sigmoid and TanH are widely used in classification but have also a disadvantage: Towards the end of the functions (in the Sigmoid more as in the TanH, see Figure 5.2 and Figure 5.3) the y -values respond less to changes in the x direction. That means that the gradient in this region tend to be small and therefore can not make a significant change in the Backpropagation.

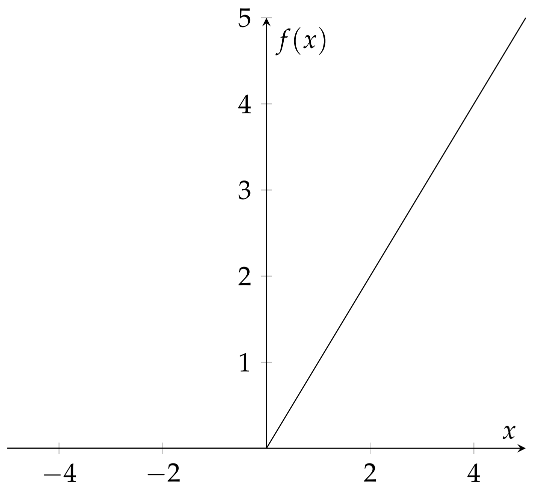

Figure 5.5: The rectified linear unit function, ReLu

Furthermore the network refuses to learn further or is very slow.[76]

Rectified linear unit, short ReLu is the state-of-the-art method which is most of the time used in current Convolutional Neural Networks. It was first introduced by Hahnloser et al[22] and is defined through the positive part of its argument. Equation 5.9 and Figure 5.5 show the calculation and the function of ReLu, where x is the input of the activation function and $f(x)$ the output.

$$f(x) = max(0, x) \tag{5.9}$$

ReLu is nonlinear, and has a range between zero and infinity, which gives a very sparse variety of output values. All values of $x < 0$ are cut off and therefore the neuron fires not for them. This means the network is lighter and does not cost as much as every neuron fires a certain value (in Sigmoid). But as all of the activation functions ReLu has also a disadvantage which represents itself in the region, where $x < 0$: The gradient in this area tempt to be zero, furthermore that means neurons which come to that state will stop responding to variations in error / input and are therefore dead. This is called the dying ReLu problem.[14]. There are also further developments on ReLu. For example Elu[12], LeakyReLu[41], which solves the gradient problem, or NoiseReLu[50].

For this work the basic ReLu activation function was used for every layer in every network. In the framework of Keras this layer is easy to add. Listing 5.2 shows how to add this layer to the network.

Listing 5.2: Activation function ReLu

```
model.add(Activation('relu'))
```

5.2.3 Loss Function

For improving the network it is necessary to quantify how bad the actual network performs. For training the network patches are used where the likelihood output value is known upfront. After pulling the patch through all layers of the Convolutional Neural Network an output prediction ends in the output layer of the network. This output prediction diverges from the value that is expected for the patch. Both values in combination tell how bad the network performs in the current state. This can further quantified via a Error - function or Loss - function. Furthermore the Loss - function is used by the optimizer (see next Subsection) for adapting the filters in the convolutional layers and improving the whole network.

The goal of the network is to minimize the Loss - function and therefore minimize the badness of the network. Over the years different Loss - functions were found by researchers for different applications. For classification tasks Binary Cross Entropy [44], Negative Log Likelihood [4] or Soft Margin Classifier [8] are used for Loss -functions. In regression tasks the Absolute Error Function [15] or Mean Squared Error [81] is used. The following equation shows the calculation of the Mean Squared Error is presented, where the vector $\hat{y}$ represents the predictions and the vector y is the observed output value.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{5.10}$$

The goal of the training of the network is to minimize the result of the Loss - function. Until the Loss (especially the Loss on the testing and validation data) decreases training more epochs makes sense. When the loss converges the optimal configuration was found an further improvements via training more epochs does not help to improve the performance of the network and therefore decrease the Loss.

For this work, the Mean Squared Error was used for quantifying the error of the network.

5.2.4 Optimization and Backpropagation

The output value of the defined Loss - function should be minimal. This can be reached via optimizing the filter kernels in the convolutional layers in a certain way. The direction and value where to decrease or increase the values in the filters is defined by the Optimization function. In Paragraph 5.1.1 the way of optimizing Multilayer perceptron networks was presented. Via gradient descent the weights (kernel values) got optimized with respect to the Loss - function.

Different Optimizers got developed over the time for faster and more accurate predictions of the different weights. In addition to the First Order Optimization Algorithm (Gradient Descent), Second Order Optimization Algorithms where used which benefit fromt the Hessian Matrix [2] for minimizing the loss. The second is more accurate than the first but highly expensive in terms of calculation time (calculation of the second order derivative)

The standard batch gradient descent (in the implementation of the MLP) perform one update for the whole dataset that means it tend to be very slow, hard to converge and tend to not fit into the memory for big datasets. A solution for that is the Stochastic Gradient Descent [5], short SGD, where an update is performed for each training sample. The problem with SGD is, that with the high frequency of updates and generated fluctuation it is complicated to find the optimal minima. Remedy brings the Mini Batch Gradient Descent approach, where the update is not performed on each training sample, but on a batch of training samples which can be defined upfront. (See Subsection 5.5.2 for the direct implementation in this work). With the help of the mini batches it reduces the variance in the parameter updates and is therefore a lot faster than updating with every training example. This is the base state - of - the - art optimizer for convolutional neural networks. The following equation show how the SGD is implemented, where η is the learning rate (see next Paragraph) and Θ is the parameter to minimize and $Q_t(\Theta)$ is the value of the loss function at the t-th mini batch.

$$\Theta_{t+1} = \Theta_t - \eta \nabla Q(\Theta_t) = \Theta_t - \eta \sum_{i=1}^{t} \nabla Q_i(\Theta_t)/t, \tag{5.11}$$

In the next paragraphs some additional options and parameters to improve the performance of the SGD (the mini patch variant) are presented and state - of - the - art derivatives are listed[78].

Learning rate The learning rate [69] controls the size of the update for each step. A higher learning rate leads to faster converge but the risk to not find the optimal minima is more high. For some optimizers a decay of the learning rate is implemented, which means that the learning rate begins with a higher number and after training a defined number of epochs it gets lower and lower. Therefore in the beginning big steps are done, but the fine grading for the optimal minima is done with lower and lower learning rates.

Momentum[63] Same as in physics momentum is used to accelerate the finding of the optimal minima in the SGD. The last update $\Delta\Theta$ is taken into account for the calculation of the new difference of the parameters. A linear combination of the last update and the actual gradient leads to faster prediction. The following equation shows the new update process.

$$\Delta\Theta_t = \alpha \Delta\Theta_{t-1} - \eta \nabla Q_i(\Theta_t) \tag{5.12}$$

$$\Theta_{t+1} = \Theta_t + \Delta\Theta t \tag{5.13}$$

Nesterov accelerated gradient[52] There is a problem with using momentum in SGD: Same as in the physical term momentum, when a ball runs down the hill the speed of the ball gets higher and higher, which means in mathematical and optimization terms, that an update can overshoot a minima and fail in finding the optimal one. Yuri Nesterov found a solution for this phenomena (therefore it is called Nesterov accelerated gradient), where the jump which is taken with the momentum is done on the previous update, than calculate the gradient and make the correction afterwards. This leads to a more dynamic update process and therefore more accurate steps to the optimal minima.

The idea of Nesterovs dynamic updates brought researches to the idea of applying this also on the learning rate, which means that the learning rate is dynamic too and lead to develop the following adaptive optimizers:

Adagrad[19] The learnng rate is dynamical, which means that it makes big updates for infrequent parameters and small update for frequent parameters. That means each parmeter has its own learning rate which makes searching for the optimal decay of the learning rate irrelevant. Adagrad stands for adaptive gradient. Since the learning rate is permanently decreasing, it could be that at a certain point no updates were performed any more and the network does not continue to learn.

Adadelta [86] It solves the mentioned problem of Adagrad with the permanently decreasing learning rate with adapting the learning rate based on a moving window of gradient updates.

Adam [34] Makes use of all the previous optimizers. Adam (which stands for Adaptive Momentum Estimation) calculate adaptive learning rates as in Adagrad and Adadelta and in addition uses again momentum in an adaptive way. That means each parameter has „its own“momentum for reaching the minima. The first moment as mean moment (also from RmsProp [75]) and the second moment (the uncentered variance) are used to update each parameter. The following equation shows the update process, where m_t is the mean moment and v_t the variance moment.

$$\Theta_{t+1} = \Theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t \tag{5.14}$$

Adam is the state-of-the art optimizer for Convolutional Neural Network so it was decided to use it for all the calculations. As a starting learning rate 0.001 was chosen. For quantifying the performance of the optimizers, a validation was done for the Fully Convolutional Network Models. The results can be seen in the Result Section (see Paragraph 5.6.1), where Adam performed the best mean validation loss comparing to the other optimizers.

5.2.5 Regularization and additional Functions

The term Overfitting[54] always comes in mind when someone talks about neural networks. When a network is perfectly trained on data but not generalized enough to perform also on new data, it overfitts. Figure 5.6 presents a comparison between the suboptimal states underfitting and overfitting and the robust case in the middle of the figure. In terms of neural network it means, that means when overfitting is happening the filter kernels are completely adjusted to the fed training data but can afterwards not perform on validation or training data. The reason for that lies in the

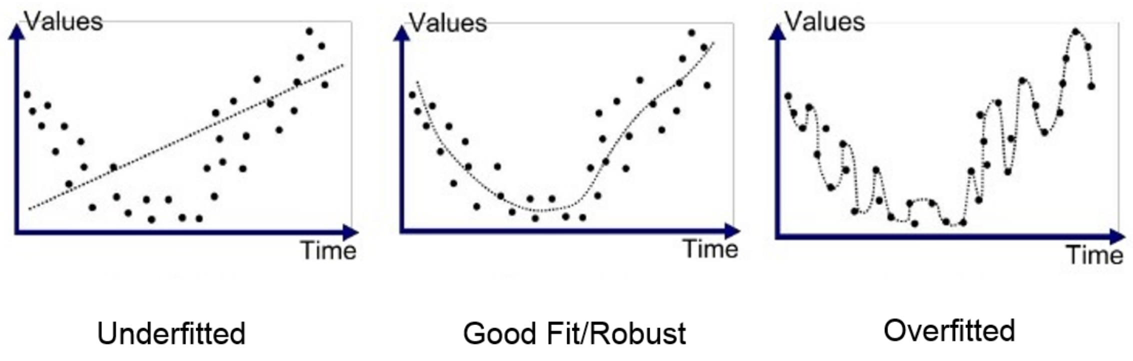

Figure 5.6: A graph which shows how a function looks like when it underfitts, overfitts or a good correlation exists in a 2D data graph [1]

input dataset and can be corrected in the model. If the input data consists of too less data or the data variation is not high enough it tends to overfitt. One possibility to get rid of that is to augment the data before processing it (see Subsection 5.4). All other possibilities are listed in the following paragraphs.

Early Stopping Validation data during the training phase shows how good the network performs on data, which it did not get trained on (see Subsection 5.5). The loss of the validation data can be used to perform Early Stopping [7]. An observation of the loss helps to interrupt the training before reaching the full amount of planned epoch steps. If the difference from the actual validation loss to the previous (epoch) validation loss falls short of a defined threshold the training stops. This saves time on the one hand, as the network does not train longer than necessary and on the other hand it saves the network from overfitting and memorizing the training data.

In the validation phase of this work, a early stopping value of 0.005 was used, therefore the network breaks up when it gets in the phase where the loss decreases not exceptionally but is also the range from where to decide if a model performs good on the validation data. In the final training phase a lower early stopping value of 0.001 was used. The networks trains longer as in the validation phase, but the value is high enough to prevent from overfitting. The `patient` value was set to two, which means, that it waits two epochs before it finally finishes the training phase after the early stopping value was reached

Dropout It regularizes the network via dropping out some of the units in the network including their connections to the previous and next layer. The selection of the units, which should be dropped is random for each iteration in the network. A probability for dropping can be set from outside. According to Srivastava et al [71] should the probability for retention in the input layers be near to the value 1.0 (more units should stay in every iteration) and for all hidden layers a value of 0.5 (50% of the units should stay in the network). A thinned network is created in every iteration in the training phase. Especially for large networks with a less amount of training data the high

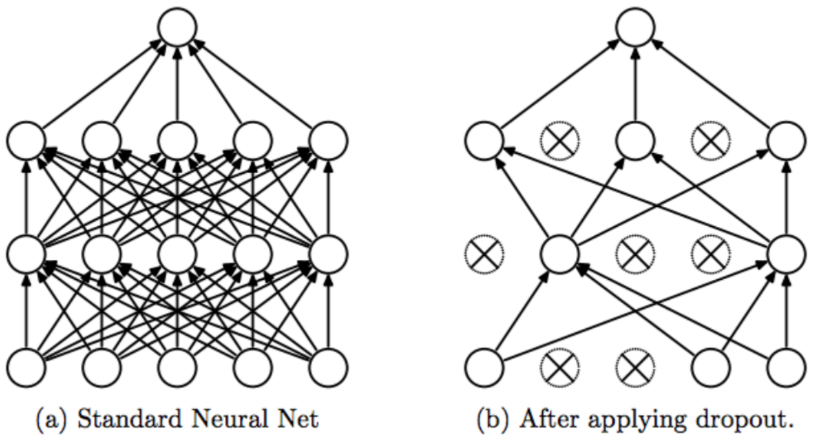

Figure 5.7: The left image shows a general neural network, the right image shows the equivalent network with applied dropout[71]

number of weights can not be trained effectively, therefore the filter in the units tend to have a lot of noise. Via thinning out the network, a better distribution happens, the filters get trained better and the network does not tend to overfit. Figure 5.7 shows a network after applying dropout, where the units with a X got dropped in this iteration.
That means in the training phase a network with n units creates 2^n possible thinned neural networks which extensive weight sharing. In the testing and prediction phase only one neural network can be used, therefore scaled down versions of the units of each thinned network are used. They get multiplied with their probability p and summed up to a single neural network.

As training data here is very sparse the Dropout was applied to every layer except for the input layer. In later models, especially in the Fully Convolutional Network models, cross validation showed that results get better without applying Dropout at all. A bigger dataset would maybe need dropout but in this scenario we skip this regularization

Downsampling the input patch from one main convolutional layer to an other layer is important for generalizing the network. It also speeds up the cycle time of one patch through the network. Three different methods help to shrink the size of the input patches. Max Pooling (or general Pooling), Strides in the convolutional layer as well as Max Pooing layer and Padding in the convolutional layer. The following Paragraphs describe them:

Downsampling - Max Pooling A Pooling layer shrinks down the size of the given patch and therefore steadies that only necessary information passes to the next (convolutional) layer. The most common implementation for CNNs is Max Pooling [10]. A kernel of $shape = (m, n)$ gets windowed over the input image with $strides = (m, n)$ (see Paragraph 5.2.5 the maximum of the pixel values defined as the most interesting and written two the output. Figure 5.8 shows an example how Max Pooling works with $(m, n) = (2, 2)$. A group of four pixels in the input image result in one pixel in the output image. A downsampling of two in each direction is given and the maximum values are preserved.

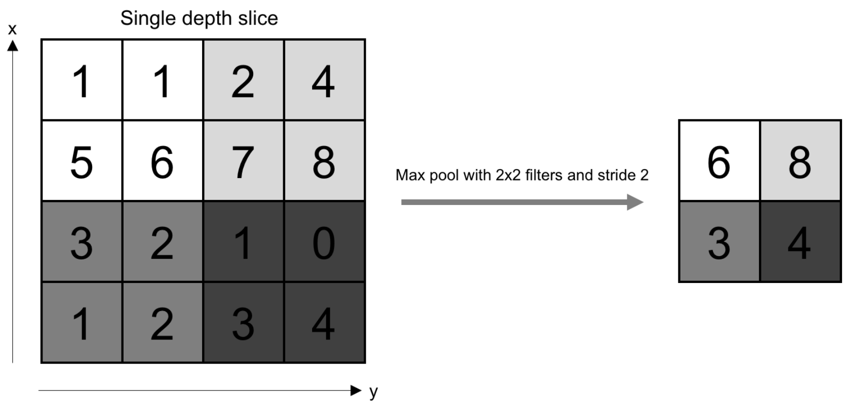

Figure 5.8: A Max Pooling of (2,2) got applied from the left image to the right image. [66]

Max Pooling is used after nearly all convolutional layers in the different models. In this work, always a shrinking of $(m, n) = (2, 2)$ is applied.

Downsampling - Strides Using Strides in convolutional layers also helps to reduce dimensionality and prevent from overfitting. With $strides = (2, 2)$ only every second pixel gets convolved with the filter kernel. As mentioned in the Paragraph above Strides are also used in combination with Max Pooling for the same reason.

Particularly in the Fully Convolutional Network Strides are used very often in the different Models because it saves a lot of computation time while the network generalizes better. In all convolutional layers where Strides is in applied, $strides = (2, 2)$ are used.

Downsampling - Padding As the previous two methods downsampled the whole image, Padding only reduces dimensionality when it is not used. Even when Strides are not used in Convolutional Networks, the input image loses pixels when it gets convolved. When $filter\_size = (x, y)$ the resulting image dimensions are $shape = (width - x - 1, height - y - 1)$. To counteract this shrinking, Padding the image can be used. Padding expands the image in every direction, so that after convolving the whole image with the filter the resulting size is again the input size.

In the patch based models Padding was not used, because a reduction of dimensionality was useful. Completely different was it in the models around the FCNs: As this Models are based on shrinking the dimensionality down and push it up again (see Model 5.3.2) the dimensions should be reproduce able via a defined factor. Padding is used in every convolutional layer, that means before convolving the filters the surrounding of the input image is padded with zeros.

Upsampling Various Downsampling methods were discussed in the previous Paragraphs. The output of the Fully Convolutional Network has the same size as the input. Through the Downsampling layers in the model, the size of the patch gets smaller and smaller. For getting the correct size back an Upsampling layer is provided. It blows the available patch up to the size of the likelihood patch.

A bilinear up sampling is used for this work. There a linear regression is done first in the one direction(x) and second in the other direction (y).

Dense layer In the Patch Based Networks a final one dimensional output has to be provided. This is done via connecting the last convolutional layer with a Dense layer. It is a fully connected layer which has the categorization classes (n - output neurons) as output or a final regression output (one - output neurons).

A combination of the `Flatten` and `Dense` - methods in Keras are used here. The `Flatten` method flattens the two dimensional patch to a one dimenional array. Afterwards the `Dense` method fully connects them to a single output, which determines the result of the regression.

5.2.6 Regression vs. classification

There are two possibilities to train and use the Convolutional Neural Network. The most common one is classification. That means the network is trained with a certain number of training data, where each input has a class label as output. After various hidden layers in the network, a fully connected layer mounts the neurons to a defined number of output layers. For example the CIFAR–10 database [36] has 60000 input patches with the size of 32x32x3 (The three layers are RGB). They are partitioned into ten different classes, such as airplane, dog, frog etc. The network calculates a probability distribution for each class. The class with the highest probability wins.

In the case of this study there are two classes: `HEAD` and `NO HEAD` . The network can be trained with different patches, which consist to either one class or the other class. The probability distribution in the end defines for each pixel (with the surrounding patch) if it belongs to the `HEAD` class or the `NO HEAD` class.

Regression is used for discreet values instead of probabilities for classes. For example to predict how much money a house would cost for different input data. It does not represent a probability for a special class but a value, which was predicted.

For this work it was decided that regression is the more suitable option. Each pixel in the input field does not get a concrete label if it is a `HEAD` or a `NO HEAD` but a value that defines how far or near it is to the head. Thus all pixels inside the head get a very high value, but the middle point of the head has the highest value. The value drops around the annotated head area.

In the implementation it means, that the output layer of the neural network is a one-dimensional value instead of two output classes. If the network should be trained in future also for other parts of the body, as the torso, legs, etc. this can be added as additional dimension in the output layer and therefore a mixture of regression and classification. More in Section 6

5.3 Models

Combining the convolutional layers, their activation functions and the regularization algorithms to a model which works best is the hard work in Deep Learning. As the trained convolutional layers are a kind of black box, one can not define a model along following different rules. In the last years challenges forced researches to develop models to classify big datasets as MNIST (handwritten digits [37], CIFAR[36], ImageNet[16], or various others. The last database, ImageNet lead to challenges, where researches duel themselves for the best performing network. From AlexNet[37] to ResNet[23] a lot of different models where created to perform better classification on the big ImageNet dataset.

This work here concentrates on a specialized case of recognition: Conversely to the image classification tasks in the ImageNet, the base for detecting the head is not only an RGB image, it is a depth frame under bad light conditions. Also it has been decided that regression is taken over classic classification (see Subsection 5.2.6). Two independent approaches of basic model structures are designed and described in the following Subsections.

5.3.1 Patch Based Network

The network gets fed via patches which got extracted from the base depth frame, it is called a Patch Based Network, short PBN. This is is the normal algorithm which is also used in classification. Patches get extracted from every annotated depth frame and they map to a certain value in the likelihood map. (More about the preparation of the database in the next Section 5.4). That means an input patch maps to one output value. In prediction mode, one patch gets put trough the network and results in a likelihood value. Afterwards a threshold or maxima extraction should decide if the patch is part of the head shape. After cross validating different Models the following two show the best performing architectures for PBN.

Model P1 This Model consists of three convolutional layers and two Max Pooling layers in between. The final Flatten and Dense layers make sure that the output consists of one final neuron. Figure 5.9 shows the full Model

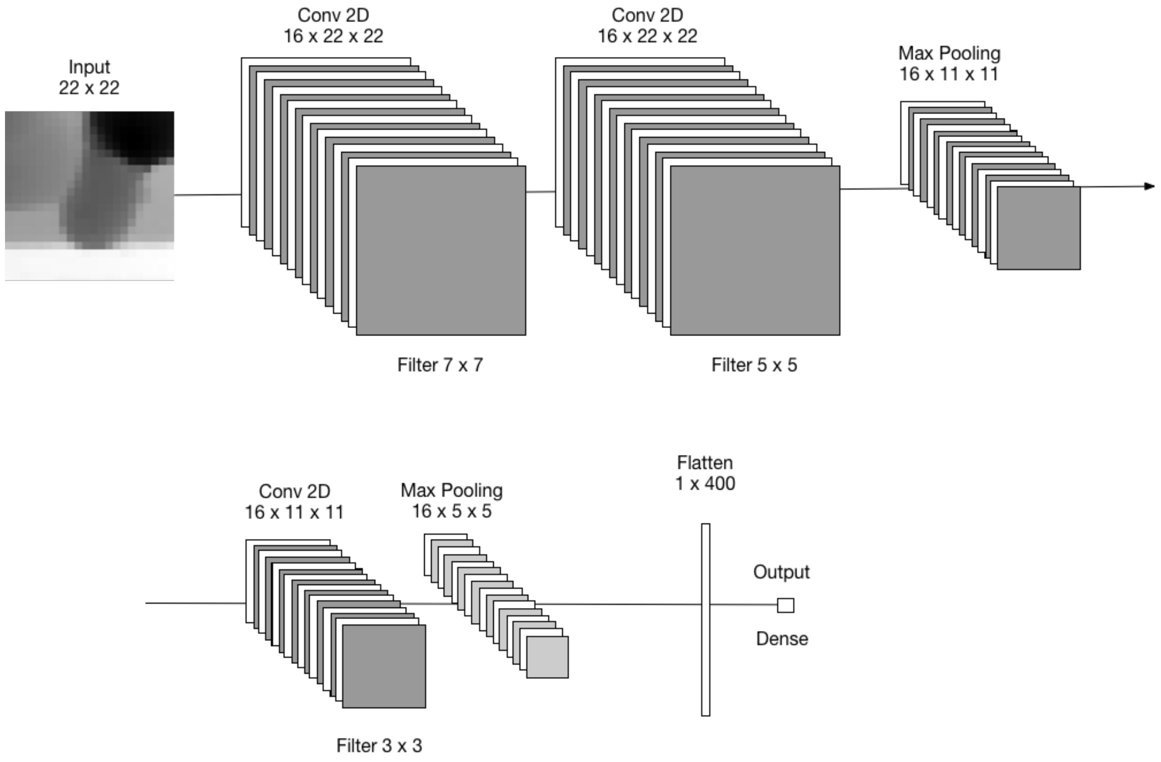

Figure 5.9: Patch Based Network - Model P1

Model P2 The second Model is an extension to the Model P1: There are four convolutional layers but their filter size is eight instead of 16. In between are three Max Pooling layers and the final Flatten and Dense layers for the single output. Figure 5.10 shows the full Model

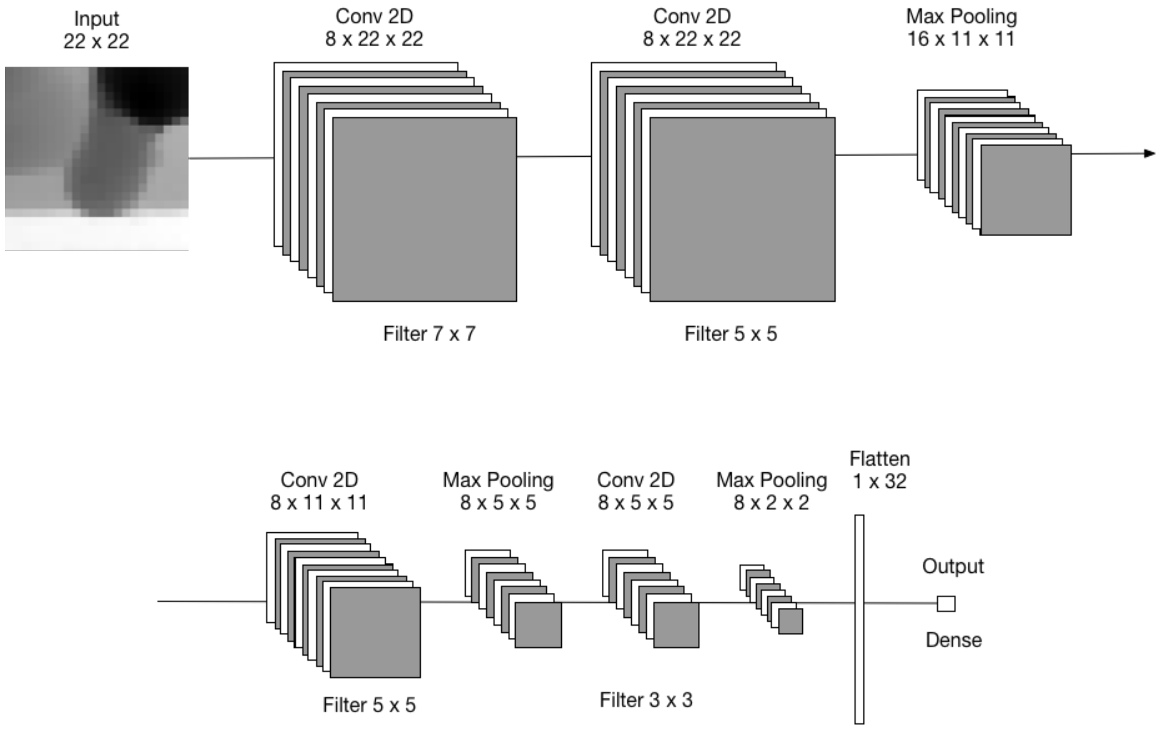

Figure 5.10: Patch Based Network - Model P2

5.3.2 Fully Convolutional Network

The second approach is called Fully Convolutional Network, short FCN, this architecture of a CNN Model was first mentioned by Long et al [40] in 2015. The basic idea is to not split up the image into different patches and therefore cut information about localization of the head in the whole image. The network gets fed by the whole image at once so that it gets broken down via convolutional layers with Strides and added Max Pooling layers and built up again via bilinear up sampling. In the end the output layer has the same dimension as the input layer, so that every input pixel maps to a likelihood and defines if the head can be found in this area or not. After cross validating over thirty different Models and investigating in the performance of them, the three different Models, which are shown below, perform best. The main difference of them is the up scaling factor, which diverges in every one of the Models and has a big impact of the final result (see Subsection 5.6.1 in Section Results).

Model F1 Four convolutional layers and two Max Pooling layers plus the final bilinear up sampling layer (up scale of factor 16) picture Model F1 in Figure 5.11

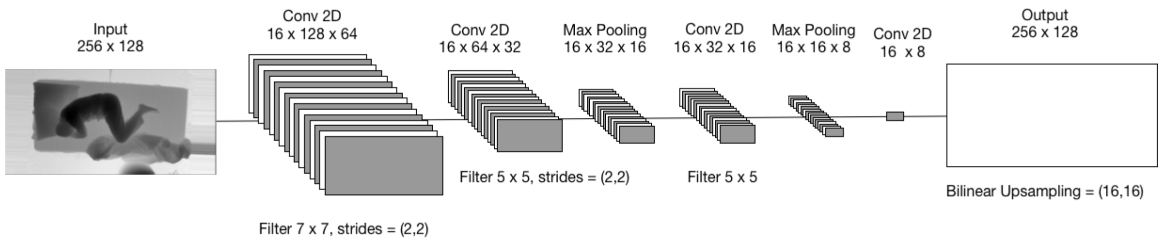

Figure 5.11: Fully Convolutional Network - Model F1

Model F2 Model F1 got extended to a more complex Model. Six convolutional layers and two Max Pooling layers plus the bilinear up sampling layer (up scale of factor 32) in the end is shown in Figure 5.12.

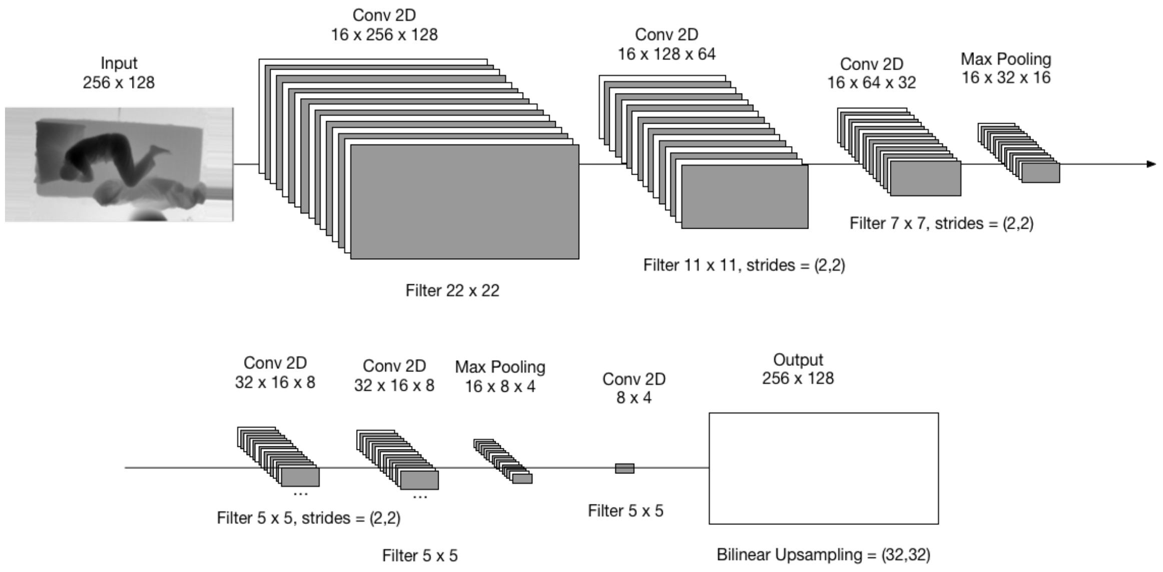

Figure 5.12: Fully Convolutional Network - Model F2

Model F3 The third FCN Model has again a higher complexity than Model F1 but does not down scale that much as Model F2. Here again six convolutional layers, two Max Pooling layers and the bilinear up sampling layer (up scale of factor eight) was used and is shown in Table 5.13

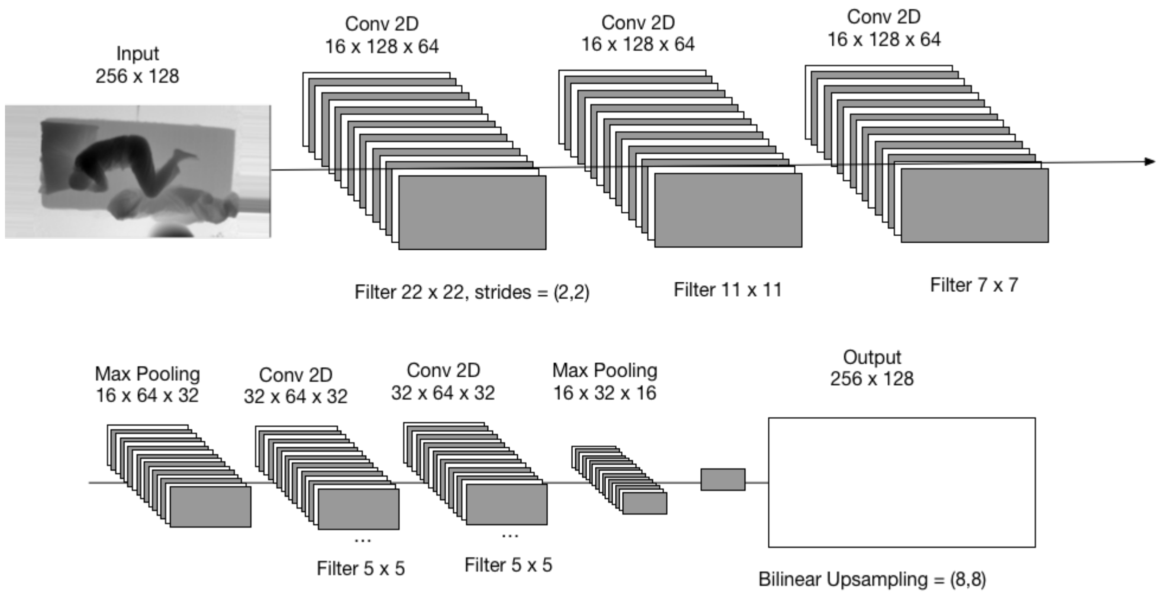

Figure 5.13: Fully Convolutional Network - Model F3

5.4 Data Preparation and Augmentation

The annotated depth frames from the second dataset (see Section 2.2.2 are used for training and testing the neural network. It is necessary to split these data into training, validation and testing data. Where the first two are used for training the data and checking the efficiency of the training and the third set of data is utilized for the end testing.

The following Subsections first present how data is selected for training and testing, then takes a look at the preparation of the input data (x - data), after that the modifications which were done on the likelihood (y - data) are presented. In the Subsections Paragraphs mark differences between data which is prepared for FCN and for the Patch based algorithms. In the end the final preparation steps for both types of algorithms are listed.

5.4.1 Data selection

Seven test persons are available of the dataset. This set got split up into training and testing persons. The first five test persons with the ids: AP280116, BK280116, CA280116, CK280116, MC280116, are sorted into the training dataset, the remaining two: ML280116 and SS280116 are sorted into the testing dataset. As each of them has approximately 200 frames annotated a rough splitting of 70% can be achieved although each individual occurs only in one of the split sets.

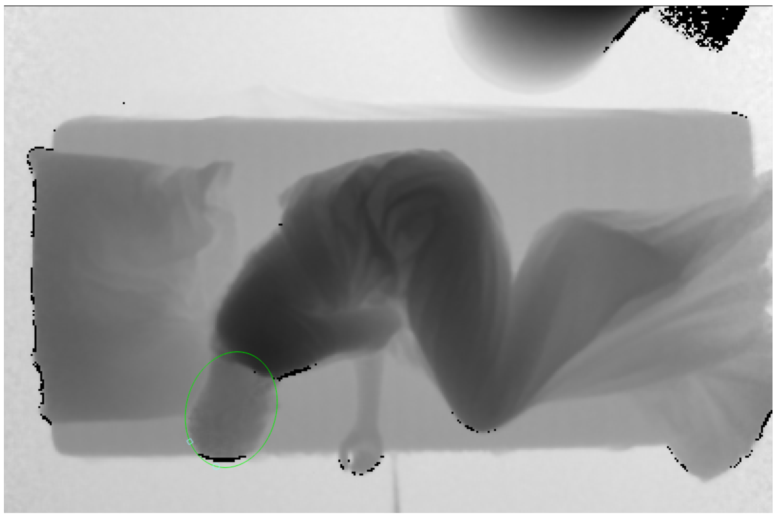

Figure 5.14: Sample of a depth frame directly from the Kinect camera with the annotated head area

Fully Convolutional Network The FCN uses the whole image to train and test at once, therefore all frames which got annotated are used. That means there are 1000 frames (Five patients times 200 frames) for training and 400 for testing.

Patch Based Network The PBN extracts a lot of patches out of the frames, since they are very often similar from the 200 frames of a patient are 100 chosen randomly for using in training and testing. That means 500 frames for training and 200 for testing.

5.4.2 x Data Preparation

Figure 5.14 shows the input depth frame which comes directly from the Kinect camera. The occasionally occurring black pixels in the depth frame are dead pixels, which can't be correctly measured from the sensor. As these pixels are noise, they are interpolated with the help of the surrounding pixels.

The python framework Scipy provides three different methods for interpolation:

- nearest
- linear
- cubic

The first is the fastest one the last the most accurate one. The amount of noise is not so high in this case and it is that important how good the original can be reconstructed therefore the first one, „nearest ", was chosen for the interpolation. Figure 5.15 shows a sample depth frame after the„nearest "interpolation. All dead pixels are interpolated and gone.

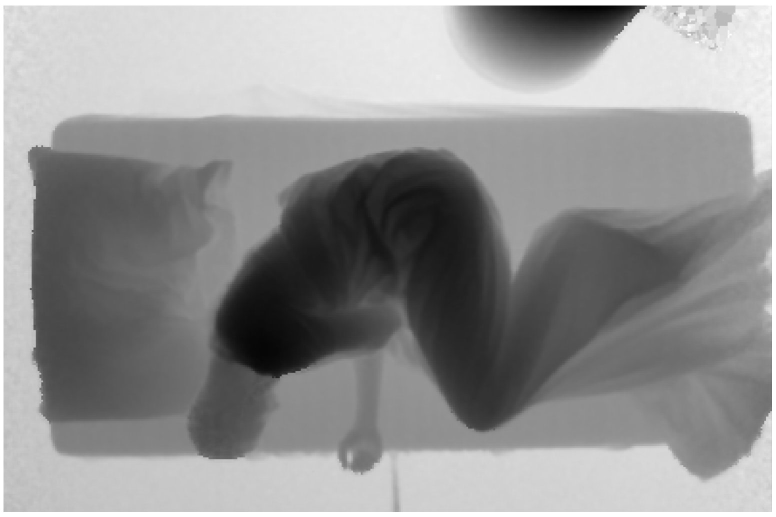

Figure 5.15: Sample of an interpolated original depth frame

Patch Based Network The depth frames have a resolution of $400px \times 264px$. To cover important surrounding information of the head (neck, pit between head and pillow, etc.) patches with at least $64px \times 64px$ (or better more) have to be extracted. The bigger the patch size is, the more convolutions have to be done during the training and execution of the network. Shrinking the frame and the patches down is an easy way to reduce convolutions and therefore execution time. A downsizing of the image by the factor of four reduces the shape to $100px \times 66px$, which allows to execute patches with $22px \times 22px$, which means that more of the surrounding area of the head can be taken into account as before (Downsizing the patch also by four would be a patch side length of 16px instead of the 22px, which are used now). The downside of losing information with the downsizing gets abolished with the fact of reducing noise and artifacts in the depth frame. (see Figure 5.16)

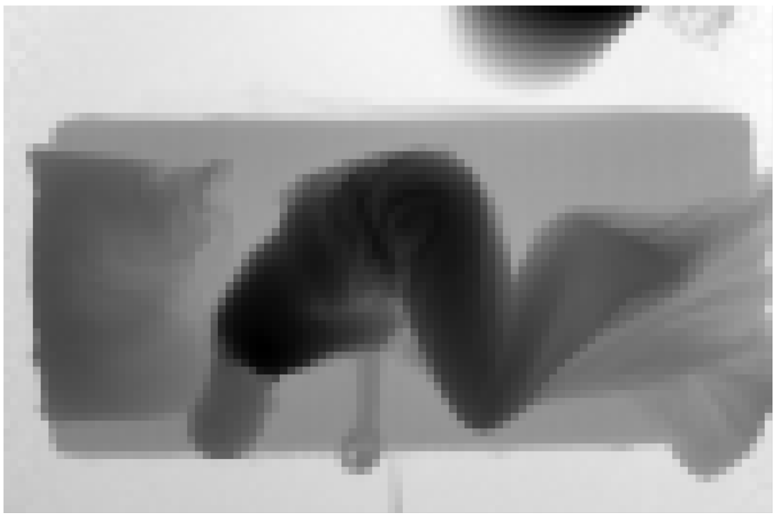

Figure 5.16: Sample of interpolated and downsized depth frame

The extraction of the patches from the depth frame is done via windowing over the whole frame - beginning from the top left to the bottom right. For every pixel in the depth frame the surrounding neighbourhood is extracted for the patch of this pixel. The extraction of the outer patches of the frame needs a padding of the image. That means the depth frame has to be stretched with half of the side length of the patch.

In this example it means that the numpy function `pad` is used to expand every edge of the depth frame with 11 pixels. Where the pixel values are copied 11 times to each side. As example Figure 5.17 shows the result after interpolation, downsizing and padding the frame.

Fully Convolutional Network The procedure for the FCN approach is similar to the PBN. After interpolating the whole image it gets downsized with the factor two to get a full image shape of $200px \times 132px$. As the image goes through a downsampling and upsampling process in the Convolutional Neural Network, the edge size should have 2^n pixels. The nearest 2^n - pixel range is $256px \times 128px$, so the whole image gets padded on the left and right side and cut on the top and bottom. Figure 5.18 shows an example of a padded, cut and downsized depth frame.

Before feeding the data into the network, additional preparation is done directly on the patch (in PBN) or the image (FCN). Subtracting all elements in the patch with the mean value of the whole patch centers the data around zero.

$$patch = patch - mean(patch) \tag{5.15}$$

According to the height of the camera above the bed, the values diverge a lot. Dividing the values of the patch by the standard deviation of the whole patch normalize them.

$$patch = \frac{patch}{std(patch)} \tag{5.16}$$

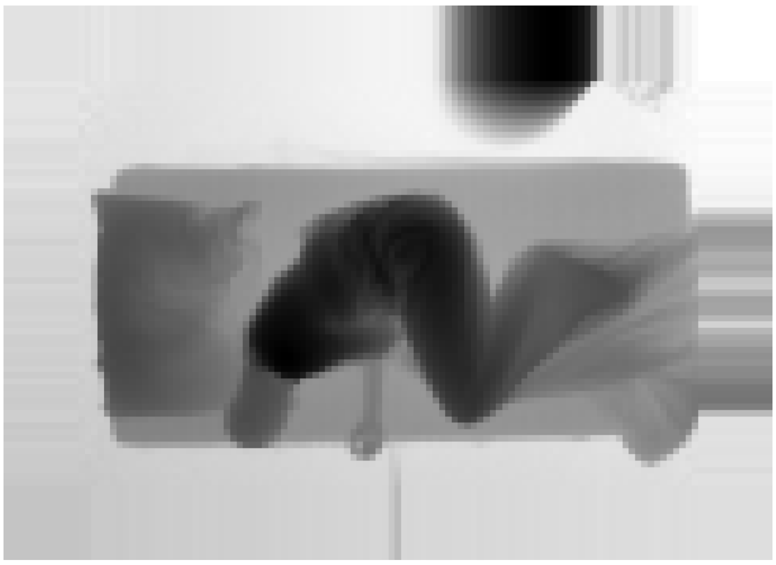
Figure 5.17: Sample of interpolated and downsized and padded depth frame for PBN

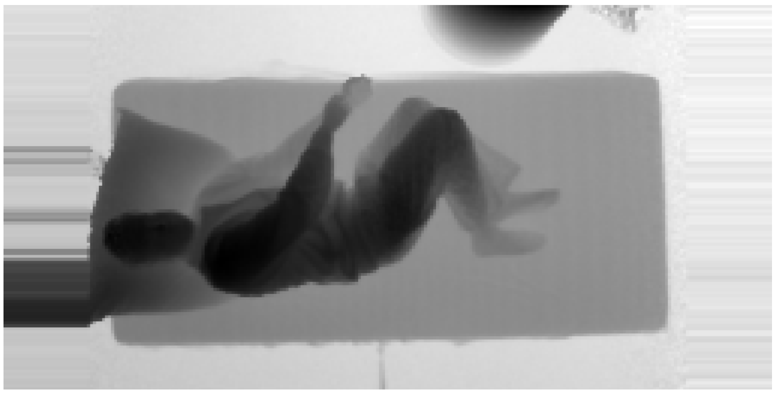
Figure 5.18: Sample of interpolated, downsized, padded, and cut depth frame for FCN

5.4.3 Data Augmentation

Data Augmentation is used to increase the dataset and variety of the dataset via slightly changing the input data. Perez et al [56] showed in their work, that with applying of traditional Data Augmentation (rotation, flip, scale) the accuracy on their validation set (Imagenet[16] dataset) increase at 4% (dogs vs goldfish classification) and 7% (dogs vs cats classification). The dataset, which is used here is sparse and has a lot of similar frames, which makes Data Augmentation more relevant. FCN and PBN have also different approaches of Data Augmentation. The following Paragraphs list the augmentations which are done to to the patches and images.

Patch Based Network The amount of patches stays the same after applying Data Augmentation but the following algorithms are applied randomly to them:

- Rotate in a degree range from zero to 20 degrees
- Shift images horizontally $0.2 \cdot width$
- Shift images vertically $0.2 \cdot height$
- Flip horizontal

Fully Convolutional Network Since the amount of training data is less then in the PBN approach (the full frame is used instead of a lot of patches in each frame) all 200 frames get extracted from each test person. The following augmentations are done to randomly selected frames:

- 100 frames are used as they are without augmentation
- 50 frames are flipped vertically
- 25 frames are rotated clockwise with a random range from zero to ten
- 25 frames are rotated counter clockwise with a random range from zero to ten

5.4.4 y Data Preparation

For training a ground truth of the expected output has to be created. As described in Section 2.2.2 there is an ellipse label for each input depth frame which represents the position (center) and shape (size of axis and angle) of the head. This information is used for building the likelihood map for the y – data.

Gaussian bell curve as likelihood method Each pixel of the input data (including the surrounding patch) maps to an output value, therefore the amount of output values is equal to the count of patches, which were extracted from the input depth frame. That means each input patch has its own output value. In this case regression is used for training the CNN (see Subsection 5.2.6, that means instead of a probability distribution for different classes, there is only one output, which defines how near or far away the patch is located to the center of the head.

A Gaussian bell curve is utilized for the likelihood method to define the value of each output pixel. The shape of the ellipse defines the appearance of the ellipse, that means the σ of the gauss in x and y direction depends on the shape of the ellipse. Equation 5.18 shows how σ is calculated, where a and b are the axis of the ellipse[43] and $fact$ defines a factor for scaling σ.

Experiments have shown, that $fact = 1$ represents the head ellipse best in the appearance. It was chosen to use this value for all the experiments

$$\sigma_x = \frac{a}{fact} \tag{5.17}$$

$$\sigma_y = \frac{b}{fact} \tag{5.18}$$

Equation 5.19 shows the calculation of the gaussian distribution, where x and y are the pixels in the patch, which is used to temporary print the values. x_0 and y_0 are the center coordinates of the patch. and $patchSize$ defines the mentioned size of the temporary patch.

$$f(x,y) = e^{\frac{-(x-x_0)^2}{2\sigma_x^2}} \cdot e^{\frac{-(y-y_0)^2}{2\sigma_y^2}}, x \in [0, patchSize], y \in [0, patchSize] \tag{5.19}$$

It was decided that the value of the $patchSize$ should be $max(a,b) \cdot 10$. This makes sure, that the full amount of values $f(x,y) > 0.0$ are printed into the patch.

After printing the gaussian distribution to the patch the whole patch can be rotated with the correct rotation of the ellipse in the annotated data. Afterwards copying the patch to a zero array of the same shape as the prepared depth frame (center of the patch is equal to center position of the annotated ellipse) finalize the likelihood calculation. Figure 5.19 shows an example.

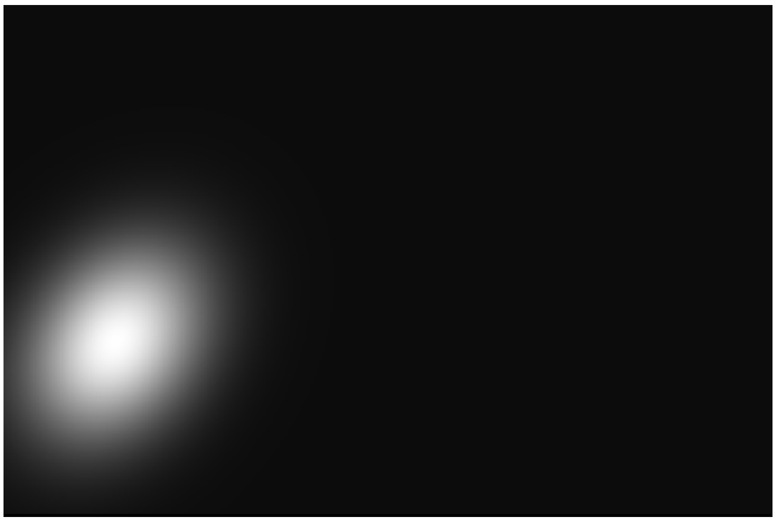

Figure 5.19: Output representation of a sample likelihood. More brighter means that the pixel is more near to the center of the head

To this point it makes no difference in the handling depending on the kind of network which is used. The final likelihood now gets further processed to patches (for PBN) or directly written to the file (FCN).

5.4.5 Data storing

The last step in the process of data extraction is to save the data in a format, which can be read from a CNN training script. HDF was chosen again because it was used for storing the input data and is fast in I/O transactions. Different file datasets get created for the different approaches. While for validation purposes two train datasets are stored in the shape of $[trainDataCount/2, 1, rows, cols]$ and one test dataset with the shape of $[testDataCount, 1, rows, cols]$ for training purposes the full training set is used, that means a shape of $[trainDataCount, 1, rows, cols]$ and the same testing set. The difference between FCN an PBN is only noticeable in the amout of entries. $rows$ and cols in the FCN approach are the downsized, cut and padded image size. In the PBN approach the patch size of the extracted patches. $trainDataCount$ and $testDataCount$ has a very low number when it come to FCN - it reflects the frame count. While in PBN it is very high, caused by the high amount of patches in every frame.

For the test dataset with the seven test persons following final dataset sizes in the proposed HDF file get reached:
Fully Convolutional Network
The amount of training data is 996. Testing data is 400, the, the shape of the input images is $256px \times 128px$ (see Subsection 5.4.2) that results in:

- Validation: $x\_train\_one = [498, 1, 256, 128]$, $x\_train\_two = [498, 1, 256, 128]$ and $x\_test =$

$[400, 1, 256, 128]$
- Training: $x\_train = [996, 1, 256, 128]$ and $x\_test = [400, 1, 256, 128]$

The likelihood data has the same data format as the input data, as their is a full pixel by pixel mapping.
Patch Based Network
The amout of training data is 827000. Testing data is 331000, the shape of the input images is $22px \times 22px$ ((see Subsection 5.4.2) that results in:

- Validation $x\_train\_one = [404000, 1, 22, 22]$, $x\_train\_two = [423000, 1, 22, 22]$ and $x\_test = [331000, 1, 22, 22]$
- Training $x\_train = [827000, 1, 22, 22]$ and $x\_test = [331000, 1, 22, 22]$

In this case likelihood data is represented via one value per input patch, that results in a one dimensional array: $y\_train\_one = [404000]$, $y\_train\_two = [423000]$, $y\_train = [827000]$ and $x\_test = [331000]$

5.5 Validation, Training and Testing

5.5.1 K - fold validation

Choosing a model and its hyper parameter is not trivial. The variation of them is endless and training the models very time consuming. Cross Validation is an important technique to evaluate if a model performs on the provided data. Kohavi et al[35] provided a technique to validate the robustness of a model. It is named K - fold validation and is based on splitting the training data in various folds. Each ith -fold of the k - folds is taken as validation data and the rest ($k \backslash i$) as training data and then training it k - times and analyzing the results. Ten - fold [59], with the most common used count of $k = 10$ would work as the following:

The training set is split into ten parts. For the first run, nine of the ten parts are trained and after each epoch validated with the tenth part. The final validation loss of this tenth part is saved when `early stopping` appears. Then another part is set aside for the validation part and a new model is trained with the rest nine of ten parts. The validation loss is saved again after each epoch. After repeating this ten times, every part was exactly one time the validation part and the ten different validation losses provide a good overview how the model performs on different data. One can build the mean value over all validation losses for concrete a value comparing with other models.

This work uses the Two - fold technique mainly caused through the lack of memory and permanent. That means each model is trained two time always with one half of the training set as training data and the other half as validation data. The results give a good overview which of the models perform best and are selected for training with the full dataset and finally validated with the separated testing dataset.

5.5.2 Training

For finally training a model with Keras and TensorFlow the training data (and validation data) has to be fitted into the prepared and compiled model. In addition to the maximum number of `epochs`, the `batch_size` of the training data has to be provided to the Keras function. The `batch_size` pretends how many patches have to be fitted to the network until a weight update is performed. Selecting the `batch_size` has an impact of the speed and performance of the training. The smaller the `batch_size`, the faster the network trains, but also the less accurate estimation of the gradient happens. A compromise has to be found.

Finding a compromise between performance and speed it was decided to take a `batch_size` of 32 for the Fully Convolutional Network and 256 for the Patch Based Network.

For validation purposes and finally training the selected model with the full training set, the dataset has to be fitted to the model. Listing 5.3 shows how this is implemented in Keras, where the `fit_generator` is used to permanently generate training data (`datagetTrain`) and validationData (`datagetVal`) from a data generator which takes the data from `x_train_one` and `x_train_two` augments them and train the model. The callbacks are added for logging the training process, early stopping and writing the best models to files.

Listing 5.3: Generating a sequential model

```
model.fit_generator(datagenTrain.flow(x_train_two, y_train_two,
   batch_size=batch_size),
        steps_per_epoch=int(len(x_train_two) // batch_size), epochs=
           epochs,
        validation_data=datagenVal.flow(x_train_one, y_train_one,
           batch_size=batch_size),
        validation_steps=int(len(x_train_one) // batch_size),
        callbacks=[json_logging_callback, checkpointer,
           earlyStopping])
```

5.5.3 Testing

Testing the models determines if they perform well on completely none seen data during the training process. Two different indicators show how good a model performs with the given test data:

The Loss Predicting a patch with a trained model leads to an output value (one single value in PBN and a two dimensional array in FCN). The difference from the predicted output and the likelihood of the testing patch can be calculated as the Loss value of this patch. Same as in the training phase, this can be done via different Loss functions.

In the training phase the Mean Squared Error was used for calculating the Loss of the training patches. In the testing phase this function was also used for calculating the testing error. The `validation_data` field in the `fit_generator` was fed with the testing data to get instant feedback how the model performs and testing error changes after each Epoch.

The Accuracy In classification Neural Networks the Accuracy is calculated via the number of correct detected patches (correct class) diverged to the full number of patches. The higher the Accuracy the better the model performs on the testing data.

Regression is used in this work, therefore a full classification can not be done. As the head regions are annotated, it is checked if the maxima in the predicted output lies in this region. When it does, it is classified as a correct detected. This can only done per frame, which means that in the PBN not every patch can be checked if is correct classified but only the full frame.

5.6 Results

The following Subsections show the results for the two base models Fully Convolutional Network and Patch Based Network.

5.6.1 Fully Convolutional Network

From the base model (see Subsection 5.3.2 different models with different hyper parameters were tested on the same training and validation set to find the best performing models and hyperparameter. The following Paragraphs show the results of different experiments.

Optimizers Different optimizers got tested based on a model with three convolutional layers, one Max Pooling layer and one Dropout layer in between and the final bilinear up sampling layer. Table 5.1 shows the results of the experiments.

Dropout For checking the impact of Dropout between the convolutional layers a Model with optimizer Adam was tested with the same dataset and various Dropout addings. The Model consists of four convolutional layers, Two Max Pooling layers in between and a final bilinear up sampling layer. The following three experiments were done and the Results are listed in Table 5.2

Experiment 1
According to the inventors of Dropout Srivastava et al [70] adding less Dropout probability to

Optimizer	Mean Validation Loss	Mean Training Loss
SGD with momentum	0.0163	0.0174
SGD without momentum	0.0180	0.0208
Adagrad	0.0169	0.0177
Adadelta	0.0161	0.0181
Adamax	0.0158	0.0170
Nadam	0.0165	0.0175
Adam	0.0155	0.0163

Table 5.1: Optimizer testing on the same Fully Convolutinonal Network model and dataset

the input layers and 0.5 to all other convolutional layers gives the best results, a Dropout of 0.2 was added to the input convolutional layer and 0.5 to the rest.

Experiment 2
The Dropout of Experiment 1 got reduced because the performance was worse than the initial Model with less Dropout. In this Experiment the Dropout layer after the input layer got kicked and the rest reduced to 0.3

Experiment 3
All Dropout layers got dropped in the Model so that no Dropout is performed on the Model.

Experiment	Mean Validation Loss	Mean Training Loss
1	0.0200	0.0206
2	0.0156	0.0163
3	0.0108	0.0107

Table 5.2: Dropout testing on the same Fully Convolutional Networks model and dataset

Cross Validation Three models, which are listed in Subsection 5.3.2, performed outstanding to the rest of the 37 evaluated models. Table 5.3 shows the validation results of 2 - fold validation. Where one can see, that Model F2 shows the best mean validation loss of them.

Model	Mean Validation Loss	Mean Training Loss
F1	0.0108	0.0107
F2	0.0098	0.0091
F3	0.0102	0.0099

Table 5.3: 2 - fold validation results of the best three models with Fully Convolutional Networks

Training The best performing three models in cross validation (see Subsection 5.3.2) got trained with the whole training set. As validation test set the test - dataset was used. Table 5.5 lists the final validation loss, training loss and time (how long one epoch trained approximately on a CPU with Dual Core Processor and on the GPU) for a training over 50 epochs

Model	Testing Loss	Training Loss	CPU Time [s]	GPU Time[s]
F1	0.0032	0.0023	33	1
F2	0.0086	0.0047	420	5
F3	0.0043	0.0015	225	3

Table 5.4: Results of a full training with the best three Fully Convolutional Network models

Testing In addition to the validation loss during the training the accuracy can be evaluated. If the finally found maxima in the predicted output has a value more than 0.5 in the according likelihood, it is counted as detected. Table 5.5 shows the detection results for the three models.

Model	Accuracy [%]
F1	99
F2	30.25
F3	89.75

Table 5.5: Accuracy of full testing with the best three Fully Convolutional Network models

5.6.2 Patch Based Network

From the base model (see Subsection 5.3.1 six different models with different hyper parameters got tested on the same training and validation set to find the best performing models and hyper parameters. The following Paragraphs show the results of different experiments.

Cross Validation Two s, which are listed in Subsection 5.3.1, got the best results in cross validation, the Results of them are listed in Table 5.6. It shows the validation results of 2 - fold validation.

Model	Mean Validation Loss	Mean Training Loss
P1	0.0117	0.0084
P2	0.0081	0,0095

Table 5.6: 2 - fold validation results of the best two models in a Patch Based Network

Training The two best performing models in cross validation (see Subsection 5.3.1) got also trained with the whole training set. As validation test set the test - dataset was used. Table 5.7 lists the final validation loss, training loss and time (how long one epoch trained approximately on a CPU with Dual Core Processor and GPU) for a training over 50 epochs.

Model	Testing Loss	Training Loss	CPU Time [s]	GPU Time[s]
P1	0.0063	0.0067	2400	149
P2	0.0060	0.0068	1380	152

Table 5.7: Results of a full training with the best three Patch Based Network models

Testing In addition to the validation loss during the training the accuracy can be evaluated. If the finally found maxima in the predicted output has a value more than 0.5 in the according likelihood, it is counted as detected. Table 5.8 shows the detection results for the two models.

Model	Accuracy [%]
1	50.72
2	57.22

Table 5.8: Accuracy of full testing with the best two Patch Based Network models

5.7 Interpretation

The following Subsections interpret the Results of training and testing the different models. First the two base architectures Fully Convolutional Network and Patch Based Network are inspected and second a summary which compares both architectures finalizes this Chapter.

5.7.1 Fully Convolutional Network

The validation steps have shown, that three models give better Results (on validation and training loss) then the rest of the 37 different models. The cross validation part of the Results (see Paragraph 5.6.2 has shown that Model F3 performs better than Model F1 and Model F2 better than Model F3. As they all got trained with the full training set it has proven, that the assumption was not completely true. Figure 5.20 shows the progress of the validation loss during the training of all three Models. One can see, that Model F1 tend to perform better than Model F3 and a lot better than Model F2 in the long run of the training. Testing the accuracy at the fully trained models shows similar results (Figure 5.21). Everything over a threshold of 0.5 means that the position is inside the shape of the head. The higher the threshold, the better the estimation to the real middle point of the annotated head.

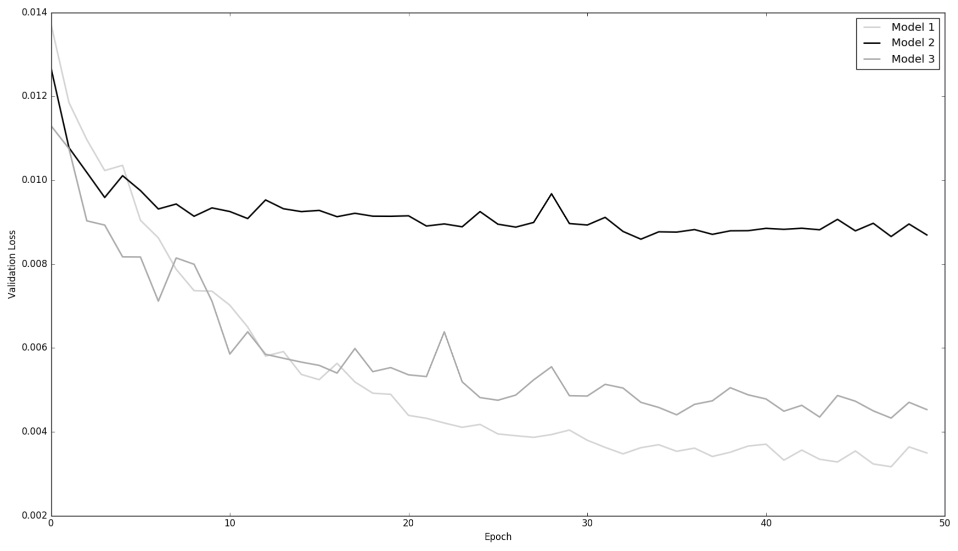

Figure 5.20: It shows the final validation loss progress on the Fully Convolutional Network models during the training of 50 epochs with the whole training dataset

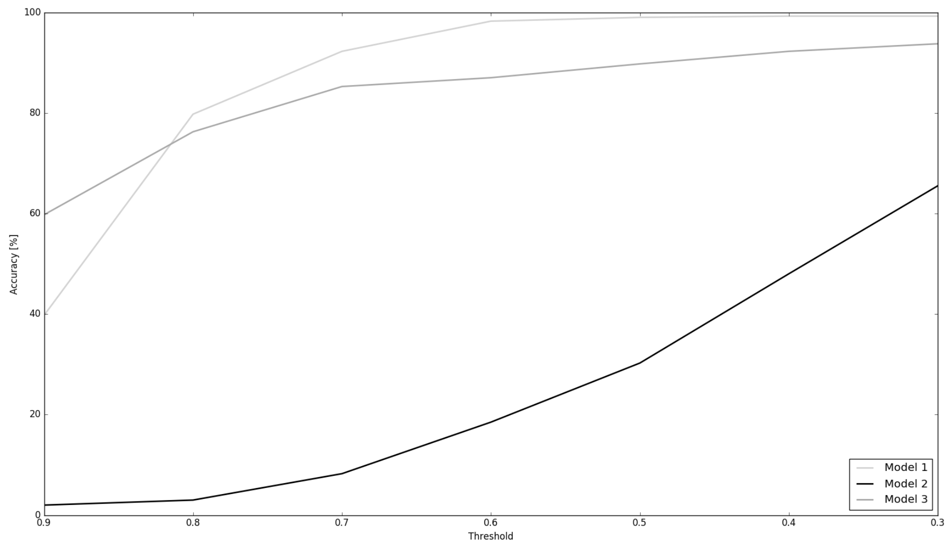

Figure 5.21: It shows the final accuracy of all three FCN models. Where the threshold gives a feeling how near the predicted maxima is away from the annotated middle point of the head. Everything up from 0.5 means that the predicted value lies into the annotated head shape.

The most surprising results gave Model F2. While the mean validation loss in cross validation performed best for this Model the training in the long run shows, that a more precise training (more data and more epochs) does not let the algorithm improve. Comparing example frames give a good answer for this: Figure 5.24 shows a negative example of detection in Model F2, where one can see that the final predicted likelihood has kind of big patches where the edges define the maxima values. Due to the fact that an up sampling of 32 times was chosen a more precise detection is not possible and a lot of head positions can not be detected. Comparing an example of Model F3, where the up sampling is eight times, in Figure 5.22 and Model F1, where the up sampling is 16 times, in Figure 5.23 shows that their the results are more fine graded. In Chapter 6 ideas are presented to overcome this problem in the future.

The behaviour of the accuracy in Figure 5.21 between Model F1 and Model F3 can also be explained with the up and down sampling of the Patches in the CNN: Where Model F3 has an up sampling layer of eight and Model F1 an up sampling layer of 16, the first performs better when it comes to the closest point of the actual head peak and the second performs better in general detection of the position. As the turn around is between a threshold of 0.8 and 0.9 the performance of Model F1 weighs a lot more, as the final results are outstanding in comparison.

Although the detection rate of Model F1 is at 99% for a threshold of 0.5 there are some problems in detecting the correct position of the head: Figure 5.25 shows an example of the main problem in failed detection, where the shoulder is detected as head instead of the real head.

5.7.2 Patch Based Network

The Patch Based Network was the initial idea to deal with Neural Networks in combination with the available dataset. Validating different Models has shown that two of them perform better than the others. These models (see Subsection 5.3.1) were trained with the whole training dataset and validated with the training set. Despite the results in the validation process the results of the training were a lot worse than the results in the Fully Convolutional Network. Compare Table **??** and Table 5.5 were the the best model in PBN has a testing loss of 0.0060 and the best model in FCN a testing loss of 0.0032 at the same training conditions.

Figure 5.26 shows the progress of the testing loss during the training of 50 epochs. One can see that the base line of reachable loss is reached after a few epochs and also the loss varies a lot in both Modes. Model P2 shows a slightly better performance of Model P2 although its model is simpler and faster in training than Model P1.

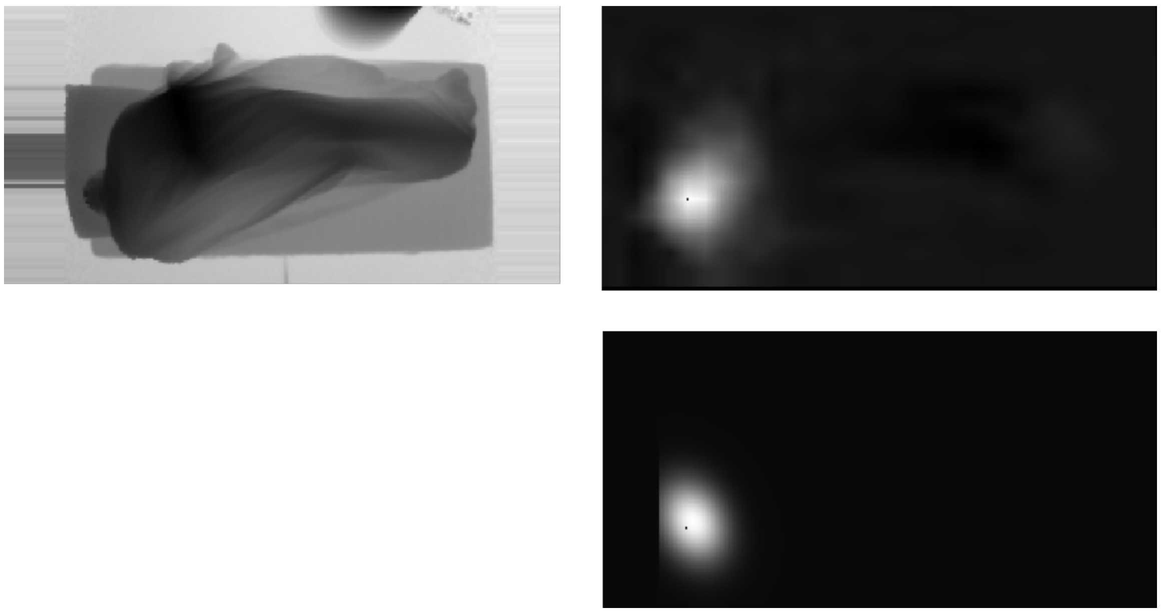

Figure 5.22: This example shows a positive head detection trained with the Model F3 network. On the top left one can see the original depth image. On the top right the predicted output is displayed and the image at the bottom right shows the original likelihood of the testing example

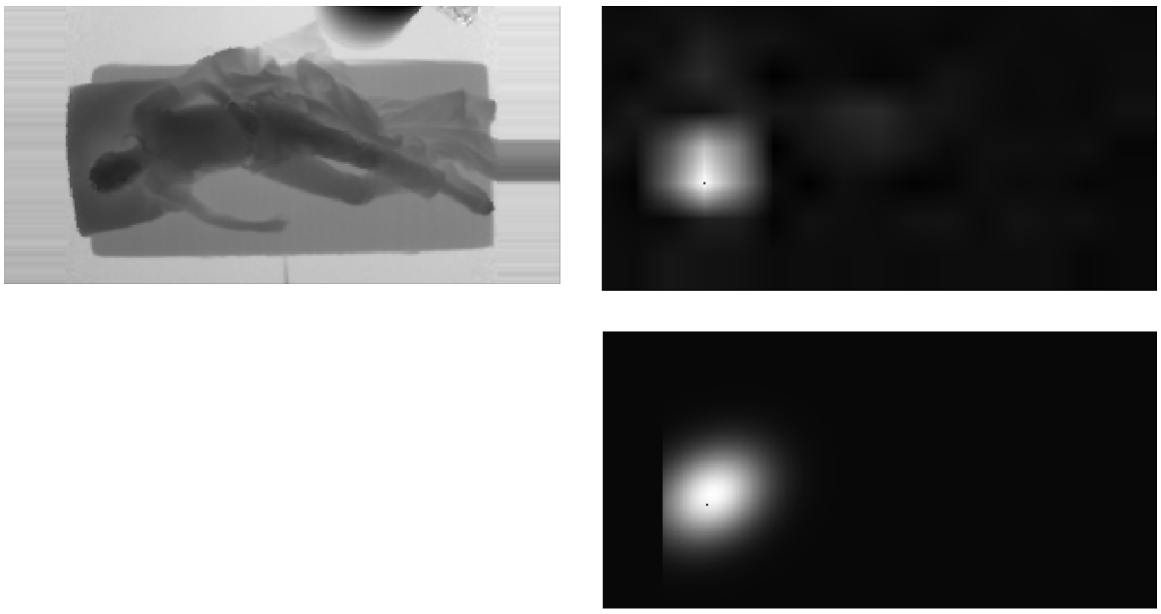

Figure 5.23: This example shows a positive head detection trained with the Model F1 network. On the top left one can see the original depth image. On the top right the predicted output is displayed and the image at the bottom right shows the original likelihood of the testing example

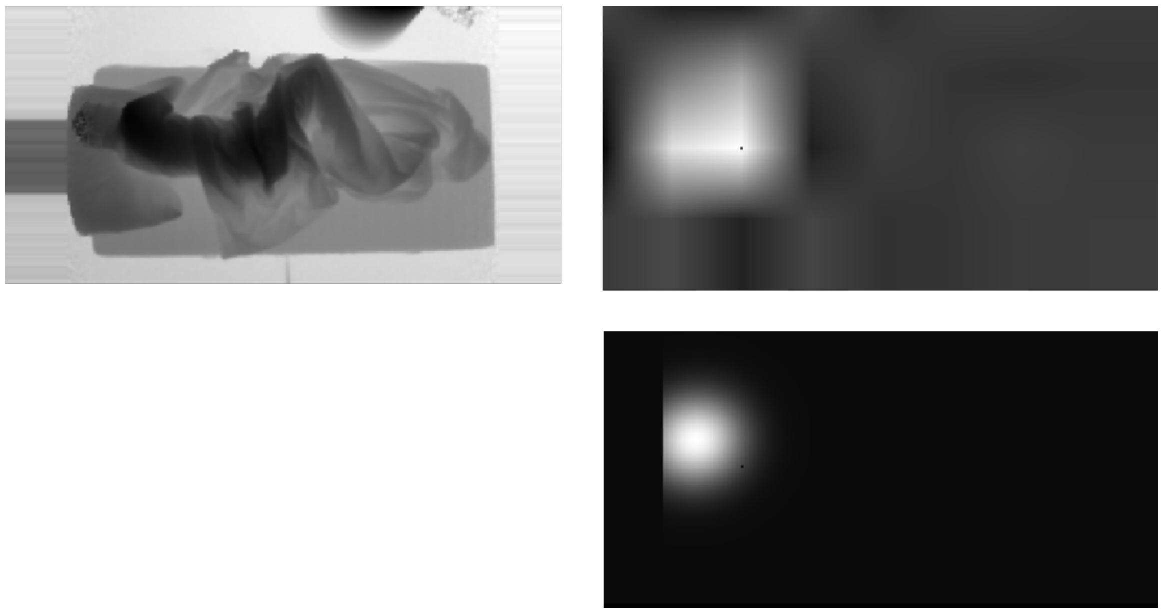
Figure 5.24: This example shows a negative head detection trained with the Model F2 network. On the top left one can see the original depth image. On the top right the predicted output is displayed and the image at the bottom right shows the original likelihood of the testing example

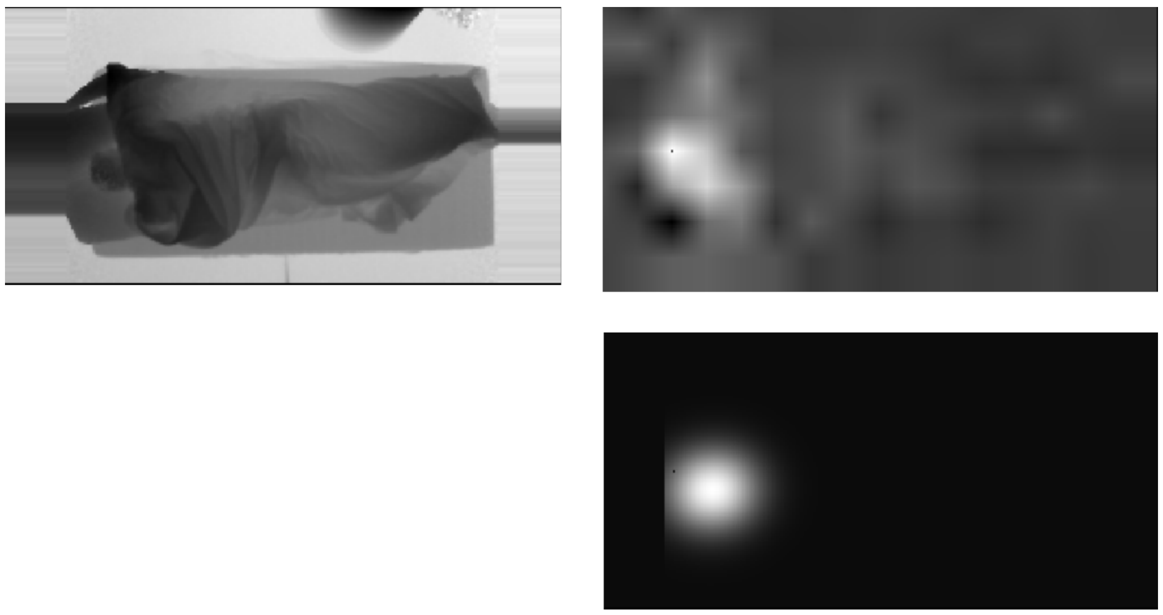
Figure 5.25: This example shows a negative head detection trained with the Model F1 network. On the top left one can see the original depth image. On the top right the predicted output is displayed and the image at the bottom right shows the original likelihood of the testing example

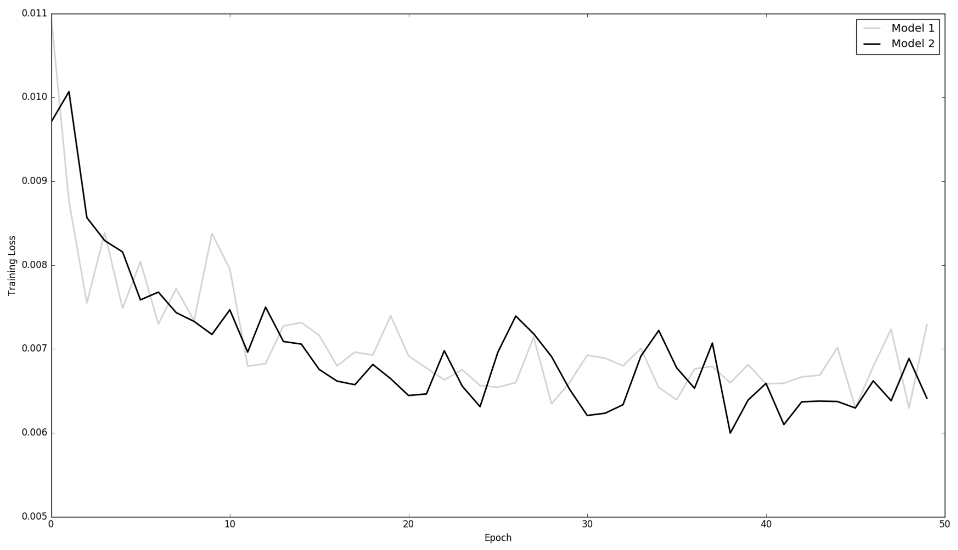

Figure 5.26: It shows the final validation loss progress during the training of 50 epochs with the whole training dataset for all PBN models

The accuracy results lie far under the results of the FCN Models. Figure 5.27 shows the trend of the accuracy over the different thresholds which except a maxima as valid detected head point. The expectations from the testing loss curve during the training are confirmed so that Model two outperforms Model one. The results of both are a lot worse than the results from the Fully Convolutional Network Models.

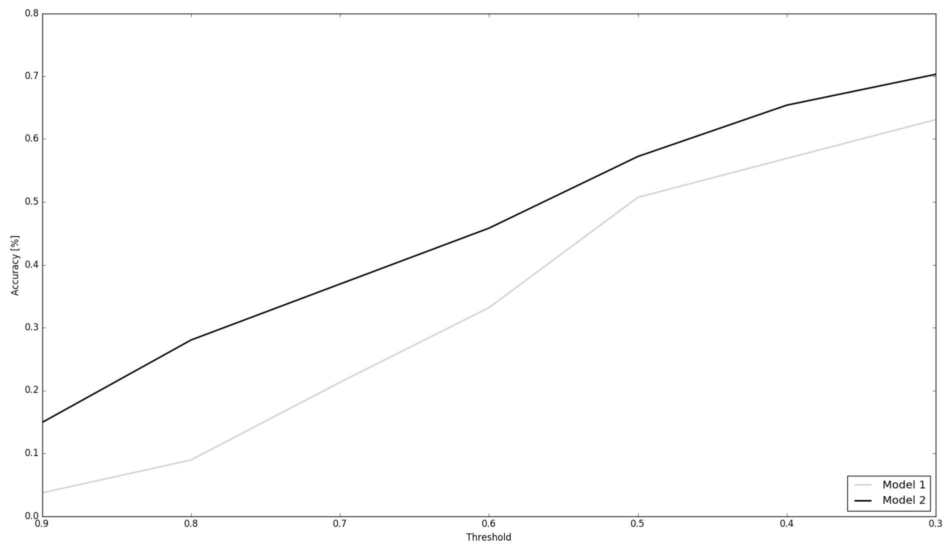

Figure 5.27: It shows the final accuracy of the two PBN models. Where the threshold gives a feeling how near the predicted maxima is away from the annotated middle point of the head. Everything up from 0.5 means that the predicted value lies into the annotated head shape.

5.7.3 Summary

The first idea of solving the head detection problem with a Convolutional Neural Network was the approach of dividing the training frames into patches (Patch Based Network) and pulling every patch through the network. The second idea was to use the full training frames and pull them as whole through the network. (Fully Convolutional Network). Looking at the results of the two base Models Fully Convolutional Network (see Subsection 5.6.1) and Patch Based Network (see Subsection 5.6.2) show that the Fully Convolutional Network outperforms the Patch Based Network in all tested models.

The better accuracy is only one advantage of the FCN. With a detection rate of 99.00% it beats the best model of the PBN (57.22%) completely and also beats the accuracy of the Two Stage Approach. Another point give the training time of the network: The size of the patches in the FCN is a lot bigger than the patches in the PBN ($256px \times 128px$ to 22×22) but the equivalent amount of training data per frame (1 patch per frame in FCN to 13200 patches per frame in PBN) in PBN results in a far slower training process. This also gave the possibility to train deeper networks in FCN than in PBN.

The FCN learns from the whole frame so it gets more informations out of the frame, where the PBN only extracts the surrounding of the head in the patch. This results often in areas which look, for the network, more as a head because the context of the whole frame is not available.

One drawback has the FCN to the PCN: The shape of the head is not detectable with this state and kind of algorithm because the down and up sampling of the frame patch results only in a heatmap where the best matching point for the head is marked. As the shape is not detectable, a gradient descent algorithm helps afterwards to detect the shape from the detected point on respectively can some ideas found in Future Work in Section 6.

All in all Model F1 gave the best testing accuracy of 99% detection rate for the position of the head. Only four frames could not be detected correctly as head positions.

6 Conclusion and Future Work

Two different algorithms were tested to detect the shape and position of a patient's head during sleep in a sleep laboratory. For both algorithms the same dataset was used. The performance of the Two Stage Approach Algorithm was good for some of the test persons but overall it did not perform in a competitive way and it only reached 83.4% accuracy on the test persons. The second algorithm, Convolutional Neural Network, is based on state-of-the-art machine learning technology and uses two available libraries, which are built on top of each other (Tensor Flow and Keras) in addition to a good data preparation. Two different base Model were tested: Fully Convolutional Network and Patch Based Network. The first gave a final accuracy of 99% for the testing data and outperforms the Patch Based Network algorithms ans the Two Stage Approach.

For future work only the FCN algorithm should be considered. The Two Stage Algorithm is very limited without a machine-learning component and the Patch Based Network is too time consuming for not getting better results The following points could be taken into consideration to improve the head detection algorithm:

More data The diversity of the frames of one person is good at this point in time. There are a lot of positions, which the people assumed during the recording (see Section 2.2.2) but the diversity of people with their habits and physical characteristics it not fully representative with seven individuals. That means in the first steps: More test persons and less images from one test person to keep the training time of the CNN as low as possible. Mixing data from test persons with real live annotated data (see dataset) for a further diversification.

Model Parameter Tuning Tweaking the model parameter can lead to better results. Especially the `Dropout` parameter should get an additional sight, when adding more training data.

Fully Convolutional Network Tuning The down and up sampling of the patches lead to a checked form of a predicted likelihood. Although Model three performed worse than Model one in testing, in the more precise area Model three (with less up sampling) performed better. The big disadvantage of the model is, that it can not determine really determine the shape of the head, which is possible with the PBN approach. Remedy would bring a further investigation in the Model building of the FCN. One proposal is, to extract the patches between the layers and mix it in the final layer, as proposed in the paper of Long et al [39]. This should give more precise informations about the shape of the head and its real center.

The CNN is currently built to learn only where possibly could be the head of the person, respectively it learns how much the probability is, that one pixel is part of the head or not. The same can be done for more body parts as feet, hands or the torso of the body. For them the same likelihood map can be created as for the head (see Section 5.4.4). In the training phase the output value is not any more represented via a single number, it contains an array where each number in the array stands for one body part. Additional layers in the model would than add more learnable filters for detecting other parts in the image. This is especially interresting for the FCN, where body part relationships are learned in the network.

Appendix

Bibliography

[1] Anup Bhande. *What is underfitting and overfitting in machine learning and how to deal with it*. Mar. 2018. URL: `https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76` (visited on 06/13/2018) (cit. on p. 33).

[2] Chris Bishop. *Exact calculation of the Hessian matrix for the multilayer perceptron*. 1992 (cit. on p. 30).

[3] Guido Borghi et al. "Poseidon: Face-from-depth for driver pose estimation." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2017, pp. 5494–5503 (cit. on p. 11).

[4] Peter AN Bosman and Dirk Thierens. "Negative log-likelihood and statistical hypothesis testing as the basis of model selection in IDEAs." In: (2000) (cit. on p. 30).

[5] Léon Bottou. "Large-scale machine learning with stochastic gradient descent." In: *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186 (cit. on p. 31).

[6] John Canny. "A computational approach to edge detection." In: *Readings in Computer Vision*. Elsevier, 1987, pp. 184–203 (cit. on p. 27).

[7] Rich Caruana, Steve Lawrence, and C Lee Giles. "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping." In: *Advances in neural information processing systems*. 2001, pp. 402–408 (cit. on p. 33).

[8] Di-Rong Chen et al. "Support vector machine soft margin classifiers: error analysis." In: *Journal of Machine Learning Research* 5.Sep (2004), pp. 1143–1175 (cit. on p. 30).

[9] Siyuan Chen, Francois Bremond, and Hugues THOMAS Hung Nguyen. "Exploring Depth Information for Head Detection with Depth Images." In: () (cit. on p. 11).

[10] Dan C Ciresan et al. "Flexible, high performance convolutional neural networks for image classification." In: *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*. Vol. 22. 1. Barcelona, Spain. 2011, p. 1237 (cit. on p. 34).

[11] Markus Clabian et al. "Head detection and localization from sparse 3D data." In: *Joint Pattern Recognition Symposium*. Springer. 2002, pp. 395–402 (cit. on p. 11).

[12] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)." In: *arXiv preprint arXiv:1511.07289* (2015) (cit. on p. 29).

[13] Ronan Collobert and Samy Bengio. "Links between perceptrons, MLPs and SVMs." In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 23 (cit. on p. 26).

[14] *CS231nConvolutional Neural Networks for Visual Recognition*. URL: `http://cs231n.github.io/neural-networks-1/` (visited on 06/07/2018) (cit. on p. 29).

[15] Arnaud De Myttenaere et al. "Mean absolute percentage error for regression models." In: *Neurocomputing* 192 (2016), pp. 38–48 (cit. on p. 30).

[16] J. Deng et al. "ImageNet: A Large-Scale Hierarchical Image Database." In: *CVPR09*. 2009 (cit. on pp. 37, 49).

[17] Ballotta Diego et al. "Fully Convolutional Network for Head Detection with Depth Images." In: *24th International Conference on Pattern Recognition (ICPR)*. 2018 (cit. on p. 11).

[18] Ciro Donalek. "Supervised and Unsupervised learning." In: *Astronomy Colloquia. USA*. 2011 (cit. on pp. 23, 24).

[19] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159 (cit. on p. 32).

[20] Richard O Duda and Peter E Hart. "Use of the Hough transformation to detect lines and curves in pictures." In: *Communications of the ACM* 15.1 (1972), pp. 11–15 (cit. on p. 27).

[21] M. Everingham et al. "The Pascal Visual Object Classes Challenge: A Retrospective." In: *International Journal of Computer Vision* 111.1 (Jan. 2015), pp. 98–136 (cit. on p. 11).

[22] Richard HR Hahnloser et al. "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit." In: *Nature* 405.6789 (2000), p. 947 (cit. on p. 29).

[23] Kaiming He et al. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778 (cit. on p. 37).

[24] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets." In: *Neural computation* 18.7 (2006), pp. 1527–1554 (cit. on p. 26).

[25] *Hospitals in Austria*. URL: `http://www.kaz.bmg.gv.at/ressourcen-inanspruchnahme/krankenanstalten.html` (visited on 04/10/2018) (cit. on p. 5).

[26] John D Hunter. "Matplotlib: A 2D graphics environment." In: *Computing in science & engineering* 9.3 (2007), pp. 90–95 (cit. on p. 9).

[27] *Introducing JSON*. URL: `http://json.org` (visited on 07/04/2017) (cit. on p. 8).

[28] IS Isa et al. "Performance comparison of different multilayer perceptron network activation functions in automated weather classification." In: *Mathematical/Analytical Modelling and Computer Simulation (AMS), 2010 Fourth Asia International Conference on*. IEEE. 2010, pp. 71–75 (cit. on p. 27).

[29] IS Isa et al. "Suitable MLP network activation functions for breast cancer and thyroid disease detection." In: *Computational Intelligence, Modelling and Simulation (CIMSiM), 2010 Second International Conference on*. IEEE. 2010, pp. 39–44 (cit. on p. 24).

[30] Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. "Artificial neural networks: A tutorial." In: *Computer* 29.3 (1996), pp. 31–44 (cit. on pp. 23, 24).

[31] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. [Online; accessed <today>]. 2001–. URL: `http://www.scipy.org/` (cit. on p. 9).

[32] Kaltenboeck. "Human Head Detection with Contour Analysis." In: (Apr. 2017) (cit. on pp. 8, 17).

[33] *Kinect for Windows SDK 2.0*. URL: `https://www.microsoft.com/en-us/download/details.aspx?id=44561` (visited on 05/15/2018) (cit. on p. 9).

[34] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014) (cit. on p. 32).

[35] Ron Kohavi et al. "A study of cross-validation and bootstrap for accuracy estimation and model selection." In: *Ijcai*. Vol. 14. 2. Montreal, Canada. 1995, pp. 1137–1145 (cit. on p. 52).

[36] Alex Krizhevsky and Geoffrey Hinton. "Learning multiple layers of features from tiny images." In: (2009) (cit. on pp. 36, 37).

[37] Yann LeCun et al. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on p. 37).

[38] Steven E Lerman et al. "Fatigue risk management in the workplace." In: *Journal of Occupational and Environmental Medicine* 54.2 (2012), pp. 231–258 (cit. on p. 5).

[39] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation." In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015 (cit. on pp. 11, 65).

[40] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440 (cit. on p. 41).

[41] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. "Rectifier nonlinearities improve neural network acoustic models." In: *Proc. icml*. Vol. 30. 1. 2013, p. 3 (cit. on p. 29).

[42] Mark W Mahowald and Carlos H Schenck. "REM sleep parasomnias." In: *Principles and Practice of Sleep Medicine: Fifth Edition*. Elsevier Inc., 2010 (cit. on p. 5).

[43] *Major and minor axe in ellipses*. URL: `http://www.mathopenref.com/ellipseaxes.html` (visited on 04/11/2018) (cit. on pp. 8, 50).

[44] Shie Mannor, Dori Peleg, and Reuven Rubinstein. "The cross entropy method for classification." In: *Proceedings of the 22nd international conference on Machine learning*. ACM. 2005, pp. 561–568 (cit. on p. 30).

[45] S Mark. "Nixon and Alberto S. Aguado. Feature Extraction and Image Processing." In: *Academic Press*. 2008, p. 88 (cit. on p. 27).

[46] WS McClulloch and Walter Pitts. "A logical calculus of the ideas immanent in neurons activity." In: *Bulletin of mathematical biophysics* 5 (1943), pp. 115–133 (cit. on pp. 23, 28).

[47] Wes McKinney et al. "Data structures for statistical computing in python." In: *Proceedings of the 9th Python in Science Conference*. Vol. 445. Austin, TX. 2010, pp. 51–56 (cit. on p. 9).

[48] Joseph J Mistovich and Keith J Karen. *Prehospital Emergency Care, 10/e+ MyBradyLab: 0-13-404506-8*. Brady/Pearson Education, 2004. Chap. Continuous Positive Airway Pressure(CPAP), p. 242 (cit. on p. 6).

[49] H Mohamed et al. "Assessment of artificial neural network for bathymetry estimation using High Resolution Satellite imagery in Shallow Lakes: case study El Burullus Lake." In: *International water technology conference*. 2015, pp. 12–14 (cit. on p. 25).

[50] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines." In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814 (cit. on p. 29).

[51] *Narcolepsy Fact Sheet*. May 2017. URL: `https://www.ninds.nih.gov/Disorders/Patient-Caregiver-Education/Fact-Sheets/Narcolepsy-Fact-Sheet` (visited on 05/15/2018) (cit. on p. 5).

[52] Yurii E Nesterov. "A method for solving the convex programming problem with convergence rate O (1/kˆ 2)." In: *Dokl. Akad. Nauk SSSR*. Vol. 269. 1983, pp. 543–547 (cit. on p. 31).

[53] Travis E Oliphant. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA, 2006 (cit. on p. 9).

[54] AB Owen. "Overfitting in neural networks." In: *COMPUTING SCIENCE AND STATISTICS* (1994), pp. 57–57 (cit. on p. 32).

[55] *Patient suffered of obstructive chronic sleep apnea prepared for sleeping services diagnostic, wearing CPAP mask covering his nose only*. May 2013. URL: `https://commons.wikimedia.org/wiki/File:Diagnostyka_bezdechu_%C5%9Brodsennego.jpg#filelinks` (visited on 05/15/2018) (cit. on p. 2).

[56] Luis Perez and Jason Wang. "The Effectiveness of Data Augmentation in Image Classification using Deep Learning." In: *arXiv preprint arXiv:1712.04621* (2017) (cit. on p. 49).

[57] *Periodic Limb Movements in Sleep*. 2016. URL: `http://sleep.health.am/sleep/more/periodic-limb-movements-in-sleep/` (visited on 05/15/2018) (cit. on p. 5).

[58] *PyQtGraph*. URL: `http://www.pyqtgraph.org` (visited on 05/15/2018) (cit. on p. 9).

[59] Payam Refaeilzadeh, Lei Tang, and Huan Liu. "Cross-validation." In: *Encyclopedia of database systems* (2016), pp. 1–7 (cit. on p. 52).

[60] *Restless Leg Syndrome*. URL: `https://www.nhlbi.nih.gov/health-topics/restless-legs-syndrome` (visited on 04/11/2018) (cit. on p. 6).

[61] Frank Rosenblatt. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Tech. rep. CORNELL AERONAUTICAL LAB INC BUFFALO NY, 1961 (cit. on p. 24).

[62] B Roth. "Narcolepsy and hypersomnia: Review and classification of 642 personally observed cases." In: *Schweizer Archiv für Neurologie, Neurochirurgie und Psychiatrie* (1976) (cit. on p. 5).

[63] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors." In: *nature* 323.6088 (1986), p. 533 (cit. on p. 31).

[64] Jürgen Schmidhuber. "Deep learning in neural networks: An overview." In: *Neural networks* 61 (2015), pp. 85–117 (cit. on p. 23).

[65] George AF Seber and Alan J Lee. *Linear regression analysis*. Vol. 329. John Wiley & Sons, 2012 (cit. on p. 23).

[66] Rajalingappaa Shanmugamani. *Deep Learning for Computer Vision by Rajalingappaa Shanmugamani*. URL: `https://www.safaribooksonline.com/library/view/deep-learning-for/9781788295628/d6fe3d14-d576-40e3-b76d-c60bc316ba80.xhtml6` (visited on 06/14/2018) (cit. on p. 35).

[67] *Sleep Apnea*. URL: `https://www.nhlbi.nih.gov/health-topics/sleep-apnea` (visited on 04/11/2018) (cit. on p. 5).

[68] *Sleep laboratories in Austria*. URL: `http://www.schlafmedizin.at/schlaflabore.html` (visited on 04/11/2018) (cit. on p. 5).

[69] Leslie N Smith. "Cyclical learning rates for training neural networks." In: *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*. IEEE. 2017, pp. 464–472 (cit. on p. 31).

[70] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: `http://jmlr.org/papers/v15/srivastava14a.html` (cit. on p. 54).

[71] Nitish Srivastava et al. "Dropout: A simple way to prevent neural networks from overfitting." In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958 (cit. on pp. 33, 34).

[72] Erik K St Louis, Angelica R Boeve, and Bradley F Boeve. "REM sleep behavior disorder in Parkinson's disease and other synucleinopathies." In: *Movement Disorders* 32.5 (2017), pp. 645–658 (cit. on p. 5).

[73] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge, 1998 (cit. on p. 23).

[74] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. "Deep neural networks for object detection." In: *Advances in neural information processing systems*. 2013, pp. 2553–2561 (cit. on p. 11).

[75] Tijmen Tieleman and Geoffrey Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude." In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31 (cit. on p. 32).

[76] Avinash Sharma V. *Understanding Activation Functions in Neural Networks*. Mar. 2017. URL: `https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0` (visited on 06/07/2018) (cit. on p. 29).

[77] Jean Vitor. *Evaluation of a Python algorithm for parallel convolution*. July 2017. URL: `http://jeanvitor.com/convolution-parallel-algorithm-python/` (visited on 06/07/2018) (cit. on p. 28).

[78] Anish Singh Walia. *Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent*. June 2017. URL: `https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f` (visited on 07/04/2018) (cit. on p. 31).

[79] Ching-Wei Wang and Andrew Hunter. "A novel approach to detect the obscured upper body in application to diagnosis of obstructive sleep apnea." In: *IAENG International Journal of Computer Science* 35.1 (2008) (cit. on p. 11).

[80] Ching-Wei Wang and Andrew Hunter. "Robust pose recognition of the obscured human body." In: *International journal of computer vision* 90.3 (2010), pp. 313–330 (cit. on p. 11).

[81] Zhou Wang and Alan C Bovik. "Mean squared error: Love it or leave it? A new look at signal fidelity measures." In: *IEEE signal processing magazine* 26.1 (2009), pp. 98–117 (cit. on p. 30).

[82] Eric W Weisstein. *Heaviside Step Function. MathWorld. A Wolfram Web Resource*. 2006 (cit. on p. 23).

[83] PJ Werbos. "New tools for prediction and analysis in the behavioral science." In: *Ph. D. Dissertation, Harvard University* (1974) (cit. on p. 25).

[84] *What is HDF?* URL: `https://support.hdfgroup.org/HDF5/whatishdf5.html` (visited on 01/21/2017) (cit. on pp. 8, 9).

[85] Lu Xia, Chia-Chih Chen, and Jake K Aggarwal. "Human detection using depth information by kinect." In: *CVPR 2011 WORKSHOPS*. IEEE. 2011, pp. 15–22 (cit. on p. 11).

[86] Matthew D Zeiler. "ADADELTA: an adaptive learning rate method." In: *arXiv preprint arXiv:1212.5701* (2012) (cit. on p. 32).