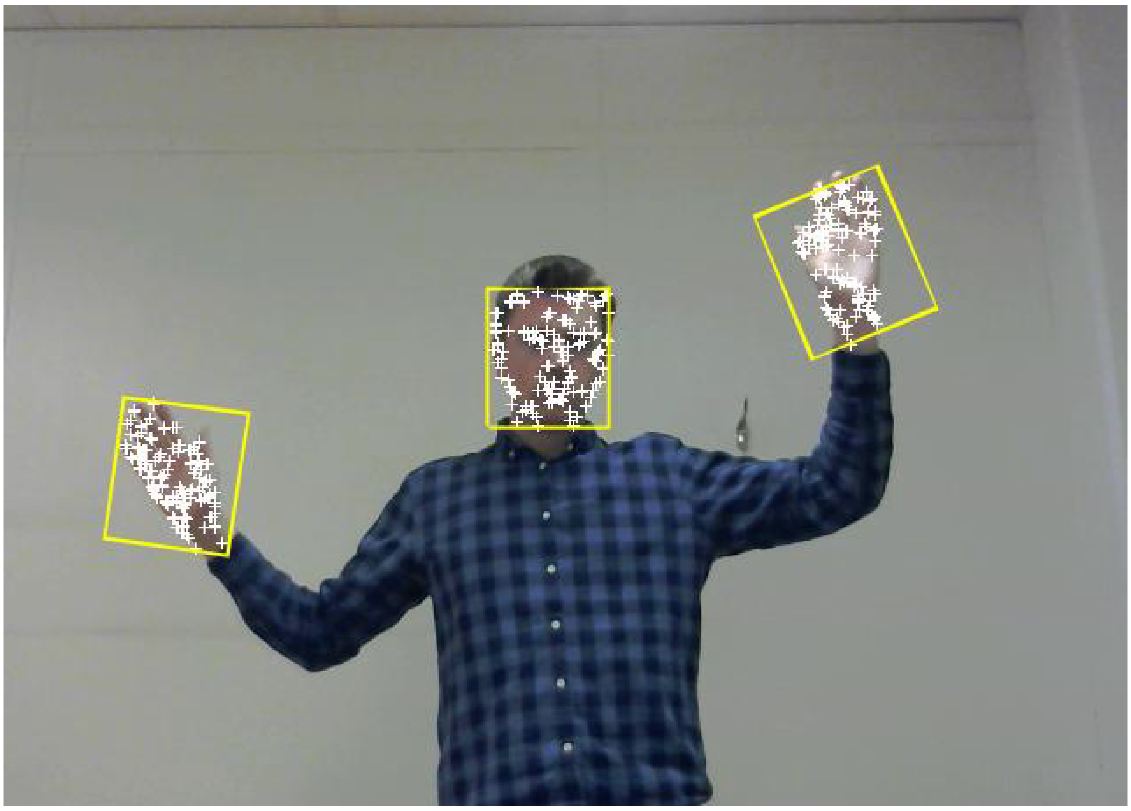
CHALMERS
UNIVERSITY OF TECHNOLOGY



Development of a motion tracking algorithm using a non-depth sensing camera to assist in rehabilitation of stroke and COPD patients

Master's thesis in Biomedical Engineering

HENRIK FRANSSON & TOBIAS PETRÉN

Department of Signals & Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis 2016:88

Development of a motion tracking algorithm using a non-depth sensing camera to assist in rehabilitation of stroke and COPD patients

HENRIK FRANSSON & TOBIAS PETRÉN

Department of Signals & Systems
Chalmers University of Technology
Gothenburg, Sweden 2016

Development of a motion tracking algorithm using a non-depth sensing camera to assist in rehabilitation of stroke and COPD patients
HENRIK FRANSSON & TOBIAS PETRÉN

Supervisor: PhD. Artur Chodorowski, Department of Signals and Systems
Examiner: Prof. Fredrik Kahl, Department of Signals and Systems

Master's Thesis 2016:88
Department of Signals & Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 (0)31 772 1000

Cover: frame from video sequence showing the tracking algorithm in action

Typeset in LaTeX
Gothenburg, Sweden 2016

Development of a motion tracking algorithm using a non-depth sensing camera to assist in rehabilitation of stroke and COPD patients

HENRIK FRANSSON & TOBIAS PETRÉN
Department of Signals & Systems
Chalmers University of Technology

Abstract

Stroke and chronic obstructive pulmonary disease (COPD) are two common diseases and the patients suffering from them are in need of rehabilitation to improve their life quality. In order to facilitate the rehabilitation process, motion controlled computer games can be used, as studies suggest. This thesis involves the development of a tracking algorithm, used in such games, able to track the hands and head of a person in real time with a non-depth sensing camera, such as those found in laptops and tablets.

The resulting algorithm consists of four main parts: background subtraction, skin extraction using the RGB and HSV color spaces, classification of hands and head with a convolutional neural network (CNN) and tracking of the classified body parts with a Kanade-Lucas-Tomasi (KLT) tracker. Running the algorithm on video recordings obtained from a common motion pattern shows that the algorithm is able to correctly track the three body parts in approximately 88.9% of the 800 frames recorded at 30 FPS and the computation time is roughly 0.48 seconds per frame. The algorithm is able to recover from a situation in which not all body parts are tracked correctly and it can handle occlusions. Furthermore, the algorithm needs some preconditions, such as good lightning and no skin-colored background or clothes, to fully function.

It is concluded that the current algorithm can potentially be used for motion controlled games. However, the current implementation is too slow to function in real time and it requires some preconditions. Implementing the algorithm in a faster language than MATLAB and enhancing the algorithm to discard some of the preconditions could potentially make the system usable for motion controlled games.

Keywords: rehabilitation, motion controlled game, stroke, COPD, visual tracking, background subtraction, skin extraction, CNN, KLT

Acknowledgements

We would like to thank our supervisor at Chalmers, Artur Chodorowski, for his support. Furthermore, we would like to thank Olof Enqvist for his support when the work stalled and Saudin Botonjic for his guidance in developing the CNN. Last but not least we would like to thank Alkit Communications for the opportunity to perform this work.

Henrik Fransson & Tobias Petrén, Gothenburg, August 2016

Contents

List of Figures

List of Tables

1 Introduction

In Sweden alone, around 30 000 people are affected by stroke each year [1]. Stroke is the most common physical cause to the need of long-term care and the annual cost of stroke for the Swedish society is approximately 20 billion SEK. A frequent outcome in the aftermath of a stroke is a decrease in muscle strength and muscle control for the affected. The impact that stroke currently has on the society, as well as on personal lives, must be reduced and one way to do this is through giving stroke patients access to an efficient rehabilitation process. [2], [3]

Chronic obstructive pulmonary disease (COPD) is another disease, but which affects the lungs. It is also one of Sweden's most common diseases In Sweden alone, about 2 500 people die yearly as a direct cause of COPD and the disease is estimated to cost the society nine billion SEK every year. Clearly, this is also an issue which must be addressed in order to reduce the impact that it has on society. One way to do this is, as with stroke, through rehabilitation in order to slow the progress of the disease. [4]

Currently, most rehabilitation for both stroke and COPD patients is done by performing exercises at a physiotherapist [2], [5], [6]. However, patients do perform their exercises at home as well [7]. Although this home-based rehabilitation has quite low adherence due to it being dull and it might also lack efficiency in due to it being unsupervised [7]. In order to decrease the amount of travelling to a physiotherapist as well as increase the adherence there is a need for a home-based rehabilitation system that is simple to use, due to the fact that most patients are elderly, as well as enjoyable. Moreover, the system must also aid in the rehabilitation process and it must have some kind of supervision.

By using a visual based tracking system, an augmented reality environment, which the user can interact with, can be generated. This type of gaming environment has been proven to aid in the rehabilitation process as well as satisfy the patients' needs [8], [9]. As an example, research has been made using a Kinect-based system in rehabilitation of stroke patients. The researchers used four different types of games, three controlled by the torso and one by upper limb movement. The results from this study was that, out of 40 patients with a mean age of 63, 93% found it enjoyable, 80% helpful and 88% thought it was something they would want to include in their therapy. Furthermore, improvements was seen in several functional outcome measures amongst the patients. [10]

Alkit Communications (hereinafter referred to as Alkit) is a Swedish research and development company currently taking part in research to create rehabilitation environments for stroke and COPD patients. It is also the company at which this thesis has been realized. Alkit is developing a system using a Kinect camera and a screen to create a gaming environment for the patients to rehabilitate in. The games in this environment are meant to be played as physical exercises and they are constructed to mimic the exercises established by physiotherapists. To accomplish this the Kinect tracks the movement of the patients body while the screen displays the patient in a virtual environment. Different objects will then appear on the screen and the patient is to interact with these objects by moving his or her body. The Kinect will, by tracking, determine if, for example, the patients right hand has the same position as the object and if this is the case a new object will appear at a different location. For example, the objects appearing could be flowers while the virtual environment could be a garden hence a simple game of picking flowers has been created. The idea behind this research is to enable home-based rehabilitation. By creating a gaming environment, patients will feel more entitled to perform the exercises and thereby do them more often, as studies suggest [8]–[11]. Furthermore, in this system the physiotherapists are able to see their patients during their exercise sessions and from the sessions results are registered in order for the physiotherapists to supervise their patients.

However, this system is relatively expensive, quite complex to install and difficult to relocate. Therefore, Alkit wants to develop a cheaper and more mobile alternative. Because of the abundance of laptops and tablets, which are commonly equipped with at least one non-depth sensing camera, it would be of great interest for the company to use them to provide a similar rehabilitation environment. Using this alternate solution would be cheaper and also more mobile. This master's thesis aims to research the tracking algorithm for such a solution and develop a prototype of it.

1.1 Purpose

The purpose of this thesis is to, by using a single non-depth sensing camera, create an algorithm able to track the human body in real time. This thesis can be seen as a proof of concept to find out if such an algorithm can be realized on a tablet or computer. Tracking the entire body without depth knowledge is difficult, thus the aim is to only track the hands and the head of the body. Since most of the games in the current system uses the position of the hands and head, tracking these three parts will be sufficient. If the algorithm is proven to be successful, it might aid in the rehabilitation process of primarily stroke and COPD patients.

The algorithm developed in this thesis needs to be robust as well as accurate. The final algorithm should therefore be able to correctly detect the location, within an error range of a few centimeters, of all the wanted limbs, with an accuracy of at least 90%, meaning that in 90% of the frames the correct body parts should be tracked. Furthermore, the aim is that the algorithm should, in real time, be able to follow the movements of the hands and head without losing track of them. Since there will be no rapid movements, due to the fact that the intended patients are quite slow in their movements, it should be sufficient that the algorithm has a maximum latency of 0.5 seconds.

The two key question that this thesis should answer are as follows.

- By using a single non-depth sensing camera, how can one accurately and rapidly track the movement of the hands and head?
- Can such an algorithm achieve similar results as the algorithm used for tracking in the Kinect?

1.2 Scope

The motion patterns that the developed algorithm focuses on tracking are shown in Figure 1.1. As seen, the algorithm should be able to detect and track all hand and head movements in the 2D plane perpendicular to the camera. Since a non-depth sensing camera is used, movements outside this 2D plane is projected onto the plane. Occlusions, such as the leftmost pose in Figure 1.1, where one hand is hiding the other, are also handled by the algorithm.

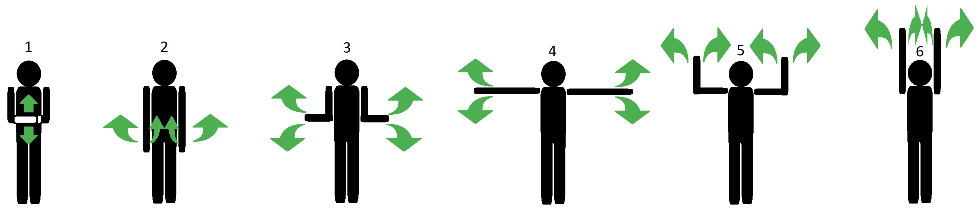


Figure 1.1: *Illustration of the motion patterns that this thesis will aim to detect.*

Before the development of the algorithm begun, some limitations were set in order to ease the development. The algorithm is to operate in a room with sufficient lightning and a static background and it does not take lost limbs into consideration, i.e. it is assumed that the person has two hands. Furthermore, the algorithm is developed for the intended use of one person at a time. The camera is to be placed on a steady object with a height of around the waist of the user. Furthermore, it should be able to record the hands while the arms are at rest as well as record the hands when they are raised above the head. The person is to be positioned in parallel to the camera, i.e. facing the camera. The algorithm does not take into consideration any position other than this. Moreover, the distance between the human body and the camera is to be roughly three meters.

1.3 Thesis outline

The thesis is outlined as follows.

Theory
This chapter starts by presenting a general overview of how a tracking algorithm is built. It then thoroughly describes the theory used in the developed algorithm and

ends with describing related work and the Kinect tracking algorithm, to give the reader an understanding of what algorithm the thesis' algorithm is being compared to.

Methods
In this chapter the different parts of the final algorithm are described and the implementation process is presented. A flowchart of the entire algorithm is also presented. The hardware and software used are also present in this chapter.

Results
The result from each part of the algorithm is presented in this chapter. Moreover, the performance of the algorithm in whole is laid out.

Discussion
In this chapter the performance of each part of the algorithm is individually discussed. The performance of the final algorithm in whole is also discussed. Finally, a discussion of future improvements of the algorithm is presented.

Conclusion
The conclusion simply concludes this work and the knowledge obtained from the work is briefly presented. Moreover, the key questions in Section 1.2 are answered in this chapter.

2
Theory

This chapter will cover all of the underlying theory used in this thesis. It is presented in sections that cover each of the four major parts of the algorithm. Furthermore, additional theory used is presented at the end of the chapter. The chapter will however begin with a brief overview of how tracking works.

2.1 Tracking overview

Following is a brief overview of the structure of a tracking algorithm in order to give the reader a general idea of this subject. It explains the most general ways of building a tracking algorithm.

Object representation

The first step in creating a tracking algorithm is to decide how the object to be tracked will be represented. Objects can be represented with their shape, appearance or both. The most common ways to represent an object's shape are by using points, primitive geometric shapes, silhouette or contour, articulated shape models and skeletal models. Figure 2.1 describes these five different approaches. Image (a) and (b) uses points, (c) and (d) uses primitive geometric shapes, (e) uses articulated shape models, (f) uses skeletal model and (g) and (h) uses contour and silhouette. [12]

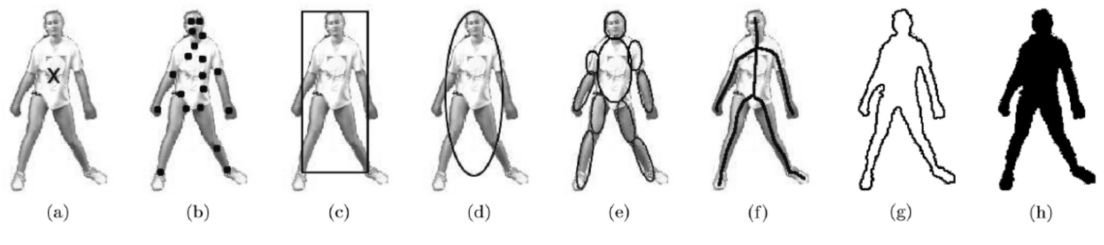


Figure 2.1: *Illustration of different object representations [12]. (a) Single point (b) Multiple points (c) Rectangular patch (d) Elliptical patch (e) Part-based multiple patches (f) Object skeleton (g) Object contour (h) Object silhouette* [12]

The most common ways of representing appearance are by using probability densities, templates, multiview appearance models and active appearance models. Probability densities extract a certain feature from a region specified by the shape model.

Templates encodes the appearance of the object from a single view. Multiview appearance models encodes different views of an object. Appearance models uses a number of landmarks extracted from the object and stores an appearance vector containing information. [12]

Features to track

When a suitable object representation has been chosen, the next step is to choose what feature or features to use for tracking the object. Below, some of the most common features to track are described.

- **Color** is a commonly used feature and there are many different color spaces that can be used, some of which are explained later in this paper.
- **Edges** represent strong changes in image intensity between nearby pixels and is most commonly used when an algorithm tries to track the boundary of the object.
- **Texture** is a representation of how the intensity in an image varies. To extract the texture from an image requires a texture descriptor, which there are a wide variety of. Two common ways of describing the texture are gray-level co-occurrence matrix (GLCM) and local binary pattern (LBP). [12]

Representation of motion

The next step is to choose how to represent the motion of the object that is being tracked. The most common ways to do this are briefly described below.

- **Uniform search** is the most basic representation in which a uniform search around the previous known position of the object is performed.
- **Probabilistic Gaussian motion** uses a Gaussian distribution as weight around the previous known position to determine the next position. The distribution gives more weight to positions close to the previous position.
- **Motion prediction** uses methods like the Kalman filter or optical flow to try and predict where the object is going to be in the next frame given the objects behavior in the previous frames.
- **Implicit motion prediction** uses optimization methods to find the object in the next frame.
- **Tracking and detection** uses an object detector together with the proposal from the motion predictor, in the form of optical flow, to determine the next position. [13]

Object detection

In order to track an object it must first be detected and located which the object detector does. Following, commonly used object detection methods are explained.

- **Point detectors** find points with expressive attribute. If these points are extracted from a specific object in an image, a feature description of this object can be created. This description can in turn be used to detect the sought object in subsequent frames.

- **Background subtraction** means subtracting the background of an image thus leaving desired objects in the foreground. This method will be further explained later in this paper.
- **Segmentation** is used to partition a given image into several regions that are perceptually distinct from one another.
- **Supervised learning** can be used both for object detection and classification. It is based on making a classifier learn different views of one or several objects from a training set so that the classifier knows how to distinguish them. [12], [14]

Object tracking

Once detected, the object should be tracked in each new frame. For this purpose, an object tracker is needed. There are several different object trackers, but most fit in one of five different groups described below as well as in Figure 2.2. [13]

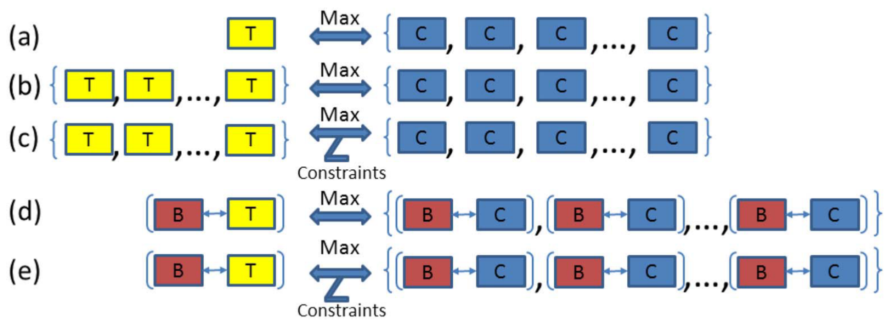


Figure 2.2: *Illustration of five different groups that describe most object trackers. The T-boxes are the representation of appearance and/or shape of the target (tracked object) for the given object tracker while the B-boxes represent background models and C-boxes represent candidate targets. (a) Matching (b) Matching using extended representation of the object (c) Matching using constraints (d) Discriminative classification (e) Discriminative classification using constraints. [13]*

The first group, (a), is the most straightforward as it uses some form of method to optimize the direct match of appearance and/or shape representation between a single model of the target and the incoming image. The second group is much like the first group, but it has several representations of the target rather than one. This is however a major difference as it allows the object tracker to have a long term memory of what the target looks. In the third group, an optimization of matches is also performed, but with constraints that have been derived from various attributes of the object, such as motion and coherence. The fourth group distinguishes itself from the three first as it uses a background model to achieve viable tracking. In this group, the object tracker seek to maximize the discrimination of the object from the background rather than by matching object representations. The fifth group is similar to the fourth, however it uses constraints (as in the third group) to determine the location of the target. [13]

2.2 Background subtraction

By creating a background model of the scene, and finding deviations in the incoming frames from this model, object detection can be achieved. The detected objects belong to the so called foreground. As an example, consider a traffic camera. The background in this scenario is the road and other static objects while the foreground is considered to be the moving vehicles on the road. Aspects to keep in mind when dealing with background subtraction are, amongst others, if the background is static or not, if there are illumination changes and if there are shadows. Furthermore, objects of interest in the foreground might temporarily become stationary. During such an event, measures should be taken to prevent these objects from merging into the background. In addition, since most background subtraction algorithms are to function in real-time, they have to be computationally efficient. Background subtraction can be divided into two categories which are described below. [12], [15]

2.2.1 Adaptive

An adaptive background subtraction adapts to changes in the background. This means that the background model constantly needs to be updated. A simple way of doing this is by using frame differencing in which the algorithm uses the image, i.e. frame, at time $t-1$ as the background model for the image at time t. Another, slightly more advanced, approach is the commonly-used median filtering. Median filtering functions by having a so called buffer with previously saved frames. This buffer consists of frames $t-n$, where $n \in \mathbb{N} \setminus \{0,1\}$, to $t-1$. From the buffer, the background model is created by taking the median at each pixel location of all the frames in the buffer resulting in a single background image. [15]

2.2.2 Non-adaptive

A non-adaptive approach does not adapt to changes in the background, hence its name. It works by, in advance, creating a background model and during the run time of the algorithm the difference between the background model and the current image is calculated. This difference is in turn thresholded (in one or several color spaces) in order to determine whether a pixel belongs to the background or not. This type of background subtraction can be seen as a version of frame differencing, but with a constant background model. [15]

2.2.3 Enhancing background subtraction

There are some basic image processing methods, presented in this section, that can be implemented in order to reduce noise and make an overall improvement of the background subtraction.

Morphology

Morphology is a technique involving non-linear operations on the shape (morphology) of parts in an image. It works by taking a binary image along with a structuring element as input and then combines these by using different set operators such as intersection and union. The structuring element (for example a 1x3 pixel area) is shifted over the image and at each pixel the elements of the structure element are compared with the set of the underlying pixels, as seen in Figure 2.3. Depending on different conditions, the pixel underneath the center of the structuring element is set to a predetermined value (0 or 1 for binary images). There are different operations in morphology of which the most common are opening, closing, erosion and dilation. What sets them apart are different set operators. Morphology is used for noise reduction as well as smoothing of edges. [16]

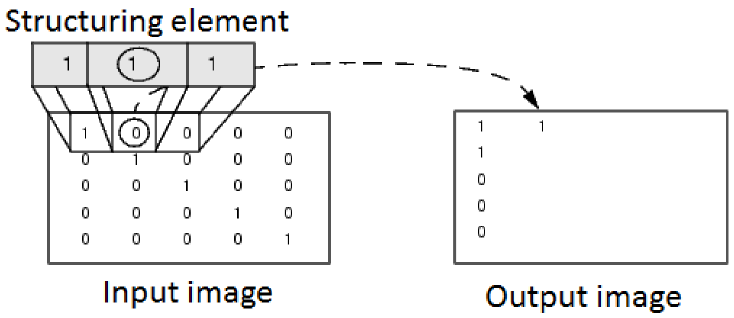


Figure 2.3: *Morphological dilation on a binary image which sets the value of the output pixel to 1. This since one of the underlying pixels has the value 1. [17]*

Median filter

A median filter is used for noise reduction in, amongst other, images. It works by running through each pixel in the image and replacing the value of each pixel with the median of the values of neighboring pixels. Depending on the application, different sizes of neighbourhoods can be chosen. [18]

Shadow removal

Another way of improving background subtraction is through the use of shadow removal. Shadows may distort images and cause background pixels to be misclassified as foreground pixels. Following, a solution for shadow removal in gray scale image sequences is explained. [19]

This method assumes that an efficient background subtraction has been performed on the image, but shadows still remain. The method focuses on comparing pixel-wise ratios using the background model, $\lambda(\mathbf{x})$, and every new frame, $I(\mathbf{x})$, of the image sequence. It is assumed that the intensity $I_s(\mathbf{x})$ of an arbitrary pixel $\mathbf{x}$ in a shadowed region can be approximated as a scaled version of the background model plus additive Gaussian noise, meaning that

$$I_s(\mathbf{x}) = \alpha(\mathbf{x})\lambda(\mathbf{x}) + \eta(\mathbf{x}), \quad \eta(\mathbf{x}) \sim \mathcal{N}(0, \sigma(\mathbf{x})^2) \tag{2.1}$$

in which $\alpha \in [0, 1]$ is related to the intensity of the shadow. Furthermore, the method simplifies α to be constant within a small neighbourhood $\Omega_s(\mathbf{x})$ of the shadowed region, i.e. $\alpha(\mathbf{x}) \approx \alpha$. This means that the expression

$$R(\mathbf{x}) = \frac{I_s(\mathbf{x})}{\lambda(\mathbf{x})} = \nu(\mathbf{x}) \tag{2.2}$$

where $\nu(\mathbf{x}) = \alpha + \eta(\mathbf{x})/\lambda(\mathbf{x}) \sim \mathcal{N}(\alpha, (\sigma(\mathbf{x})/\lambda(\mathbf{x}))^2)$, holds. In addition this means that the mean and standard deviation of $R(\mathbf{x})$, within $\Omega_s(\mathbf{x})$, can be expressed as

$$\mu_R(\mathbf{x}) = \frac{1}{N_s} \sum_{\mathbf{u} \epsilon \Omega(\mathbf{x})} R(\mathbf{u}) \tag{2.3}$$

$$\sigma_R(\mathbf{x}) = \sqrt{\frac{1}{N_s} \sum_{\mathbf{u} \epsilon \Omega(\mathbf{x})} (R(\mathbf{u} - \mu_R(\mathbf{x}))^2} \tag{2.4}$$

where N_s is the number of pixels in Ω_s. It is also assumed that $\nu(\mathbf{x})$ are independent normally distributed variables, which means that the following expression holds

$$D(\mathbf{x}) = R(\mathbf{x}) - \mu_R(\mathbf{x}) \sim \mathcal{N}(0, \sigma_D(\mathbf{x})^2) \tag{2.5}$$

where

$$\sigma_D(\mathbf{x})^2 = \frac{1}{N_s^2} \left((N_s - 1)^2 \sigma_R(\mathbf{x})^2 + \sum_{\substack{\mathbf{u} \epsilon \Omega_s(\mathbf{x}) \\ (\mathbf{u}) \neq (\mathbf{x})}} \sigma_R(\mathbf{x})^2 \right) \tag{2.6}$$

The model scans every pixel, $\mathbf{x}$, that has been detected as foreground and computes the standard deviation, $\sigma_R(\mathbf{x})$, and mean $\mu_R(\mathbf{x})$ in a small neighbourhood $\Omega_s(\mathbf{x})$ which is centered at pixel $\mathbf{x}$. If the pixel is in a shadowed region, then $D(\mathbf{x}) = R(\mathbf{x}) - \mu_R(\mathbf{x})$ is a normally distributed random variable. Moreover, there exists a constant $k_s(\beta)$, which is a function (ranging from 0 to 1) of the confidence level β, that satisfies

$$Pr|D(\mathbf{x})| > k_s(\beta)\sigma_D(\mathbf{x}) < \beta \tag{2.7}$$

Pixels that satisfy $|D(\mathbf{x})| > k_s(\beta)\sigma_D(\mathbf{x})$ are, at a confidence level β, most likely not associated with shadowed pixels. Although, pixels which do not satisfy the condition are neither guaranteed to belong to a shadowed region. However, if some of the neighbours of pixel $\mathbf{x}$ belongs to a shadowed region, the probability that $\mathbf{x}$ also does is increased. Therefore, let

$$S(\mathbf{x}) = \begin{cases} 0 & if \quad |D(\mathbf{x})| > k_s(\beta)\sigma_D(\mathbf{x}) \\ 1 & otherwise \end{cases} \tag{2.8}$$

which means that the pixel $\mathbf{x}$ will be presumed to be part of a shadow region if

$$I_{low} \leq \mu_R(\mathbf{x}) < 1 \quad and \quad \sum_{\mathbf{u} \epsilon \Omega'_s(\mathbf{x})} S(\mathbf{u}) > T_s \tag{2.9}$$

is satisfied. Here, I_{low} will prevent very dark pixels from being presumed to be shadows and T_s is the least number of pixels, within a neighbourhood Ω'_s, which must trigger the "shadow test", described by equation 2.8. [19]

2.3 Color extraction

Objects can be detected by extracting and isolating their color (or colors). To accomplish this there is a need to mathematically represent colors with numbers. This mathematical representation is achieved by the use of color spaces. By using color spaces, single colors can be isolated and extracted from an image by thresholding. Extracting skin colored objects has for instance been used in face detection. [12]

2.3.1 Color spaces

Three different color spaces are presented below.

RGB

The RGB (red, green, blue) color space is an additive color space meaning that its components are combined in different ways to create various colors. The three components, also referred to as channels, are represented by 256 discrete levels ranging from 0 (dark) to 255 (bright). As an example, red has the combination (255, 0, 0) while yellow has the combination (255, 255, 0). Worth noting is that the RGB color space is not very robust with changes in light conditions. [20]

HSV

A representation of the HSV (hue, saturation, value) color space is seen in Figure 2.4. Hue is the angle around the central vertical axis and it refers to the color type, such as red or green. Hue takes values from 0 to 360 but is often normalized to range from 0-100%. Saturation is the distance from the central vertical axis and it refers to the purity of the color. Value is the distance along the axis and refers to the brightness of the color. Both saturation and value ranges from 0-100%. [20]

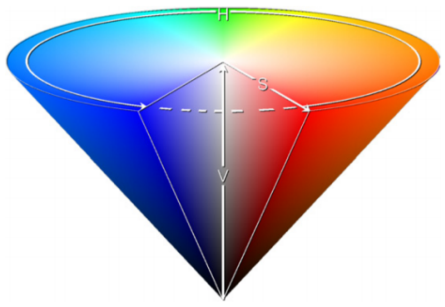


Figure 2.4: *Conical representation of the HSV color space [20].*

The transformation between the HSV and RGB color spaces is nonlinear and is defined by the following equations in which $R, G, B \in [0, 1]$, i.e. normalized values of R, G and B are used. [20]

$$H = arctan\left(\frac{R - G}{-R - G + 2B}\right) \tag{2.10}$$

$$S = \sqrt{\left(-\frac{R}{\sqrt{6}} - \frac{G}{\sqrt{6}} + \frac{2B}{\sqrt{6}}\right)^2 + \left(\frac{R}{\sqrt{6}} - \frac{G}{\sqrt{6}}\right)^2} \tag{2.11}$$

$$V = \frac{R}{3} + \frac{G}{3} + \frac{B}{3} \tag{2.12}$$

$H, S, V = 0$ when $R, G, B = 0$. The HSV color space is more sensitive to noise compared to the RGB color space [12].

YCbCr

Colors are in this color space specified in terms of luminance (the Y component) and chrominance (the Cb and Cr components). Cb represents the value of the blue color component while Cr represents the value of the red color component. The transformation between the YCbCr and RGB color spaces is linear and is defined by the following equations in which $R, G, B \in [0, 255]$. [20]

$$Y = 16 + \frac{1}{256}(65.738R + 129.057G + 25.064B) \tag{2.13}$$

$$Cb = 128 + \frac{1}{256}(-37.945R - 74.494G + 112.439B) \tag{2.14}$$

$$Cr = 128 + \frac{1}{256}(112.439R - 94.154G - 18.285B) \tag{2.15}$$

2.4 Supervised learning

Supervised learning is a subgroup of machine learning methods. It is based on learning from known datasets to make predictions about unknown data. There are two main categories of supervised learning algorithms.

- Classification algorithms that separates the data into different classes using labels
- Regression algorithms for continuous-response values

In other words, the output from a classification algorithm is some kind of discrete label while the output of a regression algorithm is a continuous number. To evaluate a supervised learning algorithm, two rates can be used. These are the validation data error rate, which is the amount (measured in percent) of misclassified validation data and the test data error rate, which is the amount of misclassified test data. The test data error rate is also referred to as misclassification rate. [21]

2.4.1 Support vector machine

A support vector machine (SVM) can be used for both classification and regression and is based on statistical learning. SVM is mainly used for binary classification, i.e. in determining between two classes, but multi-class classification using SVMs has been achieved in recent years as well. [22]

Suppose that a number of n-dimensional data points represented by features (short for feature vectors) in an n-dimensional room are given and each of these data points belong to a certain group, i.e. class. The purpose of the SVM is to, given these data (training) points, determine which class a new n-dimensional data point belongs to. In order to do so there is a need to separate the training points with a (n-1)-dimensional hyperplane. There are many such possible hyperplanes but the hyperplane that will result in the best SVM is the hyperplane with the largest margin between the two classes, i.e. the two clusters of data points in the n-dimensional room. The margin is defined as the maximal width of the space parallel to the hyperplane that contains no data points. The data points that are located on the border of the aforementioned space are known as support vectors. In Figure 2.5 an example of a two-dimensional room with a one-dimensional hyperplane separating the red and green classes is shown. [22]

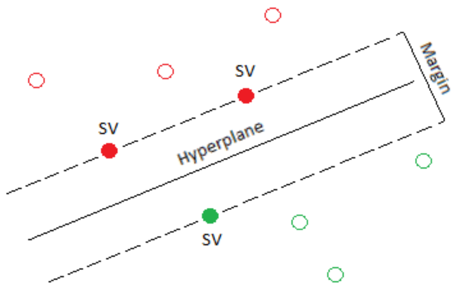


Figure 2.5: *Red and green data points in a two-dimensional room being separated by a linear one-dimensional hyperplane. Support vectors (seen as filled dots) are marked with the denotation SV.*

When a hyperplane has been found, a new n-dimensional point can be categorized into a class depending on which side of the hyperplane the point is. Figure 2.5 shows an example in which a hyperplane is able to separate the two classes entirely, i.e. it can separate all the data points. The above example is a case of linear classification, meaning that there is a simple linear hyperplane able of separating the classes. However, this is not always the case. Consider Figure 2.6 in which there are two different classes in a two-dimensional room. Clearly they can not be separated with the use of a linear hyperplane. [22]

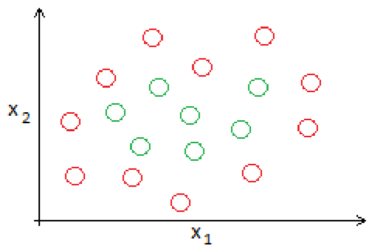


Figure 2.6: *Red and green data points in a two-dimensional room not separable with a linear hyperplane.*

To solve such a problem there is a need to create a non-linear hyperplane. One way to do this is by creating a set of polynomial functions such as those in equation

2.16. These polynomial functions will, for example, predict that a new data point at position (x_1, x_2) belongs to the green class if

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \ldots \geq 0 \tag{2.16}$$

in which $\theta_{0,\ldots,n}$ are constants obtained from training the SVM. Equation 2.16 can also be written in terms of features

$$\begin{gathered} \theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \ldots \\ \text{with the features } f_1 = x_1, f_2 = x_2, f_3 = x_1 x_2 \ldots \end{gathered} \tag{2.17}$$

However, such features quickly become complex and computationally expensive in high dimensional spaces. Therefore there is a need for better features. By using so called kernels (also referred to as similarity functions) as features, the computational load can be decreased. Consider Figure 2.7 in which there are three so called landmarks in a two-dimensional room. [23]

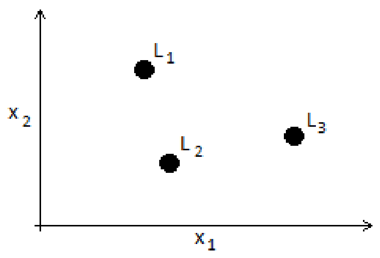


Figure 2.7: *Three landmarks, $L_{1,2,3}$, in a two-dimensional room.*

These landmarks are obtained from the training points. Given a new point (x_1, x_2) in the two-dimensional room new features can be computed depending on the points proximity to the landmarks. So given a point (x_1, x_2), denoted as a vector $\vec{x}$, features can be expressed as follows, where σ is a constant. [23]

$$f_n = similarity(\vec{x}, \vec{L_n}) = exp\left(-\frac{\|\vec{x} - \vec{L_n}\|^2}{2\sigma^2}\right) \text{ where } n \in \{1, 2, 3\} \tag{2.18}$$

The exponential term in equation 2.18 is the kernel, i.e. similarity function, and in the current form it is specifically a Gaussian kernel[1]. From equation 2.18 it can be seen that if $\vec{x} \approx L_n$ then $f_n \approx 1$ and if $\vec{x}$ is far from L_n then $f_n \approx 0$. With the use of kernels a new prediction can be made similar to that of equation 2.16, i.e. a data point at position (x_1, x_2) is considered to belong to the green class if

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0 \tag{2.19}$$

with $f_{1,2,3}$ as in equation 2.18 and $\theta_{0,\ldots,3}$ as constants. From equation 2.19 a decision boundary, i.e. non-linear hyperplane, can be created. When the decision boundary has been created it can be determined which class a new data point belongs to, therefore a classification algorithm has been created. If there is a need for a value

[1]There are of course other kernels.

determining "how much" the new data points belongs to a certain class posterior probabilites for the data point belonging to each class can be calculated. [23]

Worth noting is that when training a SVM there is a risk of overfitting it. Overfitting means that the SVM is given to many training points resulting in classes being clustered together, i.e. there is no easy distinguishing of different classes. [22]

2.4.2 Convolutional Neural Network

Artificial neural networks (ANNs) have during recent years been used more and more within the field of machine learning. These networks are, as the name suggests, inspired by how the human brain works and tries to implement this in order to recognize and classify different types of data. Convolutional neural networks (CNNs) are a type of ANN that has proven to be useful in solving difficult image recognition problems. As an example of this, a CNN developed at the New York University in the USA managed to win first prize in the 2015 fingerprint liveness detection competition, where competitors were to determine if fingerprint images were from real fingers or not, with an accuracy of 95.5% [24]. [25]

Neural networks consists of many simple processing units, called neurons, and weighted connections between these neurons. Each connection carries information between the two connected neurons where the connecting weight is either inhibitory or excitatory. Each neuron has a propagation function which receives the output of other neurons and transforms them in consideration to the connecting weights, creating a network input net, which in turn can be processed by the activation function. The activation function determines if a neuron has become activated or not, i.e. if the input given to it has caused the neuron to create such a great response (given the weighted connections) that its output is worth passing forward to the next neuron. Following are explanations of the different parts that makes up a CNN. [25]

General architecture

A CNN consists of three different major layers, convolutional layers, pooling layers and fully-connected (FC) layers. Figure 2.8 illustrates a simplified basic architecture of a CNN designed for MNIST (hand written letters) classification. [25]

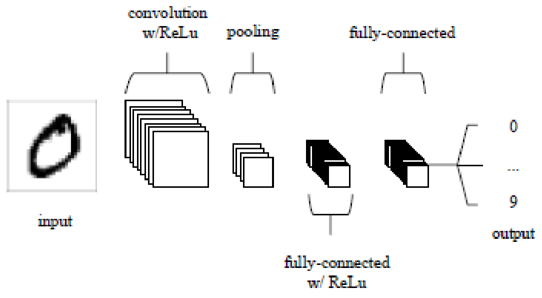


Figure 2.8: *Basic illustration of an architecture of a CNN designed for MNIST (hand written letters) classification [25].*

The input layer is simply the image to be classified, as illustrated in Figure 2.8. After this layer there is a convolutional layer which consists of a set of filters. These filters are all convolved with the entirety of the input image and seeks to produce large activation values when certain types of features appear at some spatial position in the input image. Following this, there is a rectified linear unit (ReLu) which applies an activation function to the output of the previous layer, causing some activations that were too small to be dropped. The pooling layer will then reduce the number of parameters within the activation by downsampling the output of the ReLu. Finally, the FC layer will attempt to produce scores for each class that the output can take, where the largest score will be given to the most probable classification of the input image. Figure 2.9 illustrates some example activation maps made by the first convolutional layer of the MNIST CNN on different input images, after training. It is possible to see how some of the filters try to find different characteristics for different inputs in order to have unique activations depending on the input. [25]

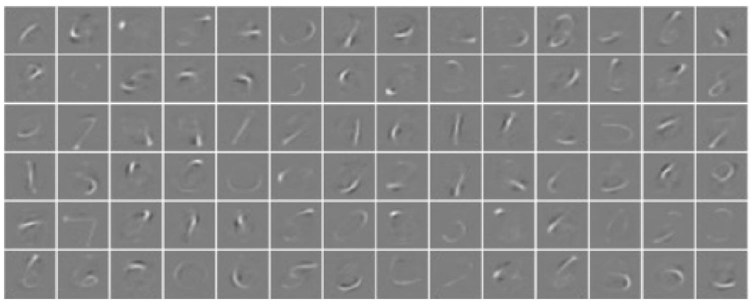


Figure 2.9: *Illustration of the activations made from the first convolutional layer of the MNIST CNN, after training. It can be seen that the network has successfully picked up characteristics that are unique to different input images. [25]*

A major advantage that CNNs have over classical ANNs is that the number of weights (filter values) used in just one neuron in is far less. For classical ANNs, for a colored 64x64 input image, this number is 12,288 (64x64x3) while for the CNNs the same number is 108 (6x6x3) if a typical 6x6 filter is used. This is due to that classical ANNs use fully connected layers. Having fully connected layers does not only mean larger computational complexity, but it also means that the neuron does not take into consideration the spatial structure of the data. In other words, pixels that are far apart are treated in the same way as those that lie close to each other. This is something that is not realistic for images in which features can be very localized. Clearly CNN is the better choice when classifying images. Figure 2.10 illustrates the difference between a convolutional layer and a FC layer. [25]

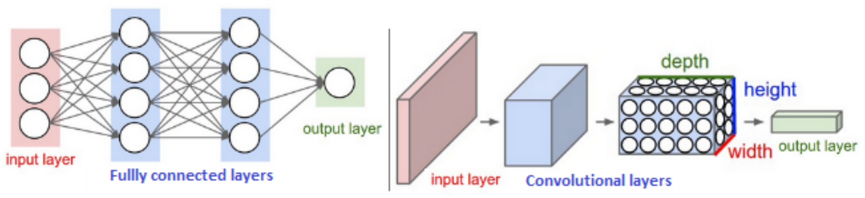


Figure 2.10: *Illustration of the difference between a classical three layered neural network and a three layered CNN. As can be seen in one of the layers, the CNN arranges its neurons in three dimensions (width, height, depth). In the example given, there is a colored image as input which means its width and height is the dimensions of the image and its depth is three due to the three color channels. Therefore, it can be seen as a 3D volume which is transformed into a 3D volume of neuron activations (illustrated in green). [26]*

Convolutional layer

The convolutional layer is the core of a CNN. It consists of a set of learnable filters which are commonly chosen to be spatially small in comparison to the input. When the input is passed through the convolutional layer, each filter is convolved across the entirety of the input creating a large 2D activation map. The activation maps from all the filters are then stacked along the depth dimension to form the full output volume of the convolutional layer. Figure 2.9 illustrates examples of what an activation map can look like. The filters are trained to "fire" when a certain type of feature is present in the input image, something that is commonly known as activations. Figure 2.11 illustrates an example of what a convolution with a filter may look like. The convolution is performed by first both vertically and horizontally flipping the filter and then placing it over the input area, marked in red. Each number in the filter is then multiplied with the corresponding number underneath. All of these multiplications (a total of 9 for this filter) are then summed together and the result is what the neuron holds on to. [25]

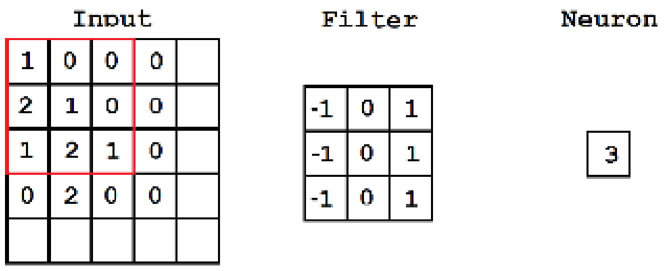


Figure 2.11: *Illustration of how a convolutional layer works. The filter in the middle is convolved with the upper left part of the input, marked in red, creating a value which a neuron holds on to, illustrated to the right.*

An advantage with CNNs is that there in general are few parameters to tune, in comparison to other feature recognition methods. This makes CNNs relatively attractive to use. The parameters that can be chosen for the convolutional layers include filter size, depth of output volume, zero-padding, weights and stride. The depth parameter is chosen depending on the depth of the input data . So if the input data is an RGB image, the depth of the convolutional layer will be three. Decreasing the depth will reduce the number of neurons that are connected to the same region of the input. Although this reduces computational complexity, it also reduces the pattern recognition capabilities of the layer. Zero padding is, like the name suggests, the adding of zeros around the border of the input volume. This parameter is chosen in order to control the size of the output volume, but also to guide the filters to treat the entire input data equally, including the borders. The weights, which is the part of the filter that will be taught and successfully improved during the training of the network, should be initialized with care for deep (multi-layered) networks. They should be initialized smaller the more input connections a layer has as this will ensure that all activations have the same variance, i.e. no activation will only be able to find a small set of features, while others can find a large set of features. Stride is the parameter that determines how many pixels that the filter should "jump" after every filtering cycle. For example if the input is a 30x30 gray scale image, the filter size is 5x5 and the stride is 2, the first pixel that the filter will be centered around will be at location (2,2) and the second location will be (2+2,2). [25], [27]

Pooling layer

In order to further reduce the number of parameters and computational complexity of the model, pooling layers are used. They are often placed after convolutional layers and they operate over each activation map through the use of "MAX" functions. This has given them the name of max-pooling layers. They most often have the dimensionality of 2x2 with a stride of 2 along the spatial dimension of the input, preserving its depth. This reduces the size of the input to 25% of its original size. [25]

Fully-connected layer

In FC layers, all neurons are directly connected to each other, as shown in Figure 2.10. FC layers have high computational complexity, but they are needed in the end of a CNN in order to let all the processed data be reduced to a few output parameters. In order to make the output of the last FC layer readable as a posterior probability, a loss function layer of the type softmax is often used. This layer converts its input vector to a probability vector where the sum of all elements is one and each individual element lies between 0 and 1. Since each element in the input vector represents one category, the output vector of the softmax contains the probabilities to which category that the input data to the CNN belongs to. [25], [28]

Activation functions

There are two commonly used activation functions that are used to suppress certain outputs from different layers in a network. These are the Sigmoid and ReLu which are illustrated in Figure 2.12. Of the two, the ReLu is far more wider used as it has proven to be less computational expensive and stochastic gradient decent (explained later) converges faster than Sigmoid. [26]

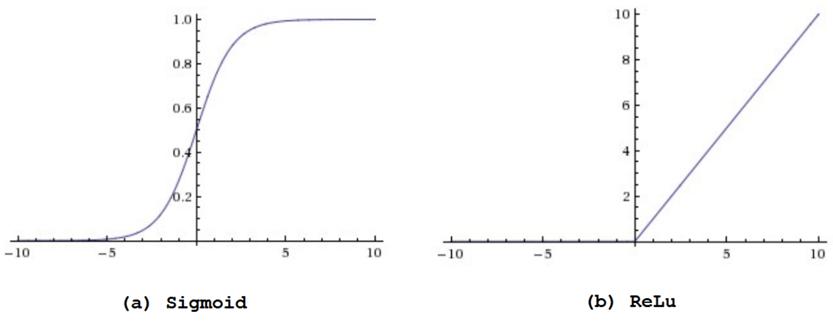


Figure 2.12: *(a) Sigmoid (b) Rectified linear unit activation functions [26].*

Dropout layer

Overfitting is a serious problem in neural networks. It is created when noise is present in the training data leading to a complicated relationships between input and output that is only present in the training data, but not in the real data. To reduce overfitting and the impact of noise, dropout can be used. [29]

In the ideal case, the best way to regularize a CNN, i.e. reduce the overfitting, would be to average the prediction of all settings of all possible parameters of the network, weighting each setting (given the training data) with its posterior probability. In other words, use all possible combinations of the parameters in the network to find the optimal value for them. Another ideal approach to retrieve a more reliable output from a CNN for a given problem is to combine several different CNNs and average their output. [29]

However, the above mentioned approaches are only applicable if we have near infinite amount of computational power and huge amounts of training data. Dropout is a method that addresses the issues with both of the ideal approaches. The method is illustrated in Figure 2.13. It drops units/neurons at random, i.e. some units input and output connections are temporarily removed. Each unit is retrained with a probability of p, which is often set to 0.5 as this is the optimal for most networks. Using dropout in the network can therefore be seen as training 2^n (where n is the number of units in the network) networks with extensive weight sharing where each of these networks rarely gets trained. The units that are removed are only removed during the training of one image, and after that the dropout is performed once again. During testing, the dropout is completely removed (p is set to 1). By doing so, it is the same as combining 2^n networks with shared weights into one network. [29]

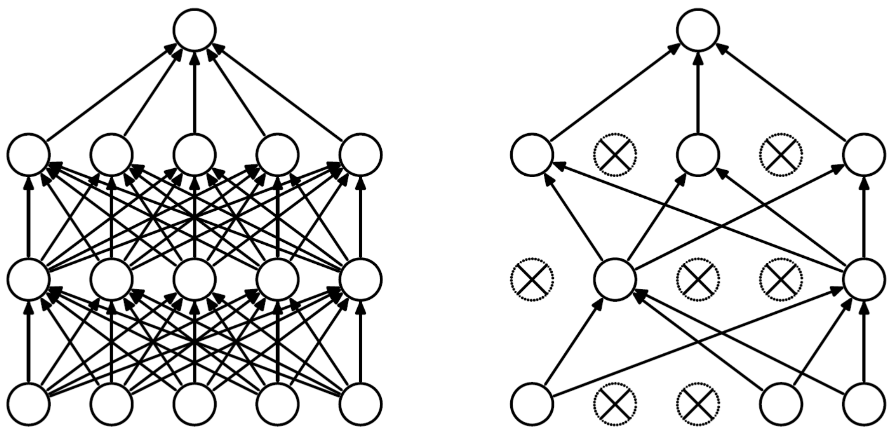


Figure 2.13: *Illustration of how dropout works. (a) A standard network that has two FC layers. (b) An example of a thinned net that has been created by applying dropout on the network in (a). Crossed units are those that have been dropped. [29]*

Training methods

Using gradient decent is a common way of training CNNs. However, there are three methods of performing gradient decent, namely the batch, mini-batch and on-line methods. In the batch training method, the changes the that weights make are accumulated over a presentation of all the training data (an epoch) before it is applied. The on-line training method on the other hand updates the weights after every presentation of each training example (instance). Furthermore, there is another alternative that is known as the mini-batch method in which the weights changes are accumulated over a set number of instances before the weights are updated. If the update frequency, often referred to as batch size, is set to one, it is the same as on-line training and if it is set to be as large as the entire training set, it is the same as batch training. Many researchers are divided between which of the three methods that is deemed as the most effective. [30]

Gradient descent

A common approach to teaching the weights after every iteration, so that they produce a better result, is gradient descent. To understand how it works, consider a network with only one weight. In order to find the best value for this weight which will minimize the loss, it is possible to test, say, 1000 different values for this weight and simply pick the value that gives the lowest validation data error rate, i.e. lowest loss value. However, doing this is rather computationally expensive and performing the same brute force search for a network with 3 weights would need to testing of $1000 \times 1000 \times 1000 = 1\ 000\ 000\ 000$ different combinations of weights, which is obviously not computationally efficient. Consider the network with one weight again, but this time two different values for the weight are tested, both relatively close to the current value of the weight, but one value being larger and the other smaller. For the resulting loss values which these different values for the weight produces, gradients can be computed. In this way, the direction in which the loss decreases can be found. By allowing the weight to step wise take a value along the direction in which the loss curve decreases, until the lowest value of the loss function is reached, is known as gradient descent. The length of the step in which the weight descends the loss curve is commonly known as learning rate and is explained more later. The problem of calculating the gradient for all weights still remains. Consider the simple regression network in Figure 2.14. [31], [32]

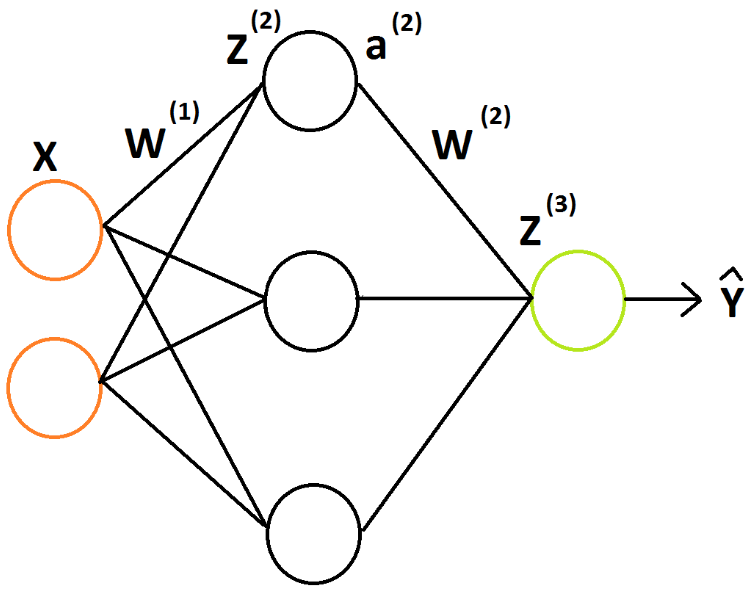


Figure 2.14: *Example network. It has an input layer, X, of size 2, FC layer of size 3 and output layer of size 1. $W^{(1)}$ denotes the weights between the input and the FC layer. $Z^{(2)}$ denotes the activity of the second layer holds the relationship $Z^{(2)} = XW^{(1)}$. At the same time, $a^{(2)}$ is the activations of $Z^{(2)}$ and holds the relationship $a^{(2)} = f(Z^{(2)})$ where $f(x)$ is the sigmoid activation function. Furthermore, $Z^{(3)} = a^2W^{(2)}$ is the activity of the third layer while $\hat{Y} = f(Z^{(3)})$ is the output of the network. [31]*

The loss/cost function is denoted as

$$J = \sum \frac{1}{2}(Y - \hat{Y})^2 = \sum \frac{1}{2}(Y - f(f(XW^{(1)})W^{(2)}))^2 \tag{2.20}$$

where Y is the real value of the output data. To calculate the gradient of J with regards to W is the same as asking "How does J change when $W^{(1)}$ and $W^{(2)}$ change?". If this gradient is negative, it means that the loss function is going downhill, and the other way around for a positive gradient. When the gradient reaches a low value, preferably zero, the global minima has been found for the loss function, and the network has been optimized. The sum in equation 2.20 is the number of training data which is used to improve the weights. Consider the situation when only one training data is used, this yields the gradient, with regards to $W^{(2)}$

$$\frac{dJ}{dW^{(2)}} = \frac{d\frac{1}{2}(Y - \hat{Y})^2}{dW^{(2)}} \tag{2.21}$$

Now, consider the chain rule which states

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \tag{2.22}$$

which means that equation 2.21 can be evaluated as

$$\frac{dJ}{dW^{(2)}} = -(Y - \hat{Y})\frac{d\hat{Y}}{dW^{(2)}} = -(Y - \hat{Y})\frac{d\hat{Y}}{dZ^{(3)}}\frac{dZ^{(3)}}{dW^{(2)}} \tag{2.23}$$

Given that $\hat{Y} = f(Z^{(3)})$ it is possible to express this as

$$\frac{dJ}{dW^{(2)}} = -(Y - \hat{Y})f'(Z^{(3)})\frac{dZ^{(3)}}{dW^{(2)}} \tag{2.24}$$

The last term in the above equation can be seen as in Figure 2.15. I.e. for each synapse/connection, $\frac{dZ^{(3)}}{dW^{(2)}}$ is the activation a on the synapse. Equation 2.25 illustrates this. [32], [33]

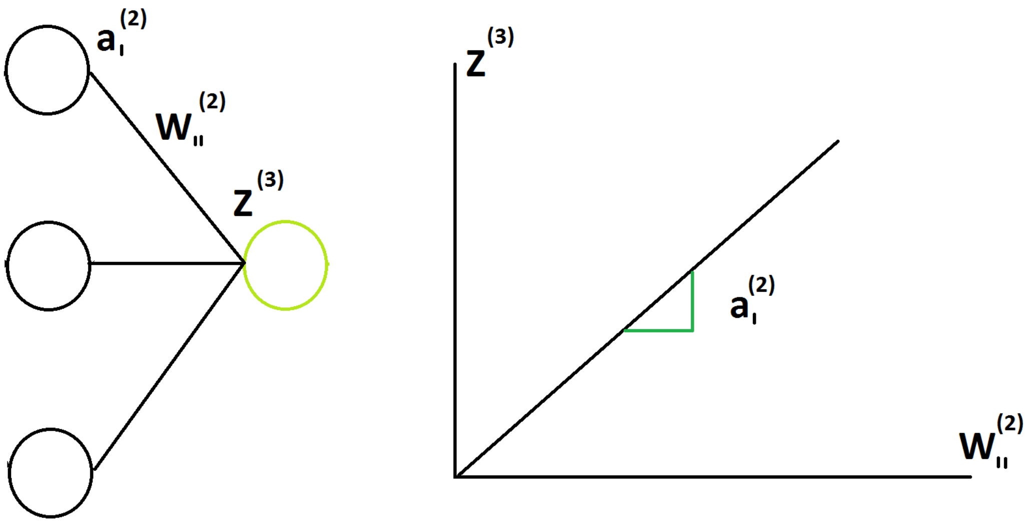


Figure 2.15: *The right part illustrates the relationship between $Z^{(3)}$ and $W_{II}^{(2)}$, where the slope is simply $a_I^{(2)}$. The left part of this figure illustrates where in the network the variables are located. [33]*

$$\frac{dZ^{(3)}}{dW^{(2)}} = \begin{bmatrix} a_{II}^{(2)} & a_{I2}^{(2)} & a_{I3}^{(2)} \\ a_{2I}^{(2)} & a_{22}^{(2)} & a_{23}^{(2)} \\ a_{3I}^{(2)} & a_{32}^{(2)} & a_{33}^{(2)} \end{bmatrix} \tag{2.25}$$

What happens here is a backpropagation of the error. I.e. the weight that contribute more to the error, will have larger activations, yield larger $\frac{dJ}{dW^{(2)}}$ values and will be changed more when the gradient decent is performed. To express this more mathematically, consider expressing $\delta^{(3)}$, the backpropagation error, as

$$\delta^{(3)} = -(Y - \hat{Y})f'(Z^{(3)}) \tag{2.26}$$

This means that it is possible to express the loss function gradient, with regards to $dW^{(2)}$, as

$$\frac{dJ}{dW^{(2)}} = \delta^{(3)}(a^{(2)})^T \tag{2.27}$$

meaning that the loss curve gradient, with regards to $dW^{(2)}$, is the backpropagation error times the activity of each synapse. This makes sense because it means, as mentioned above, that the amount of learning for each weight is based on how much it is at fault for causing an error. [33]

Consider now the first weights, $dW^{(1)}$. Following the same concept as for $dW^{(2)}$ and applying the chain rule, the following expression for $\frac{dJ}{dW^{(1)}}$ can be concluded

$$\frac{dJ}{dW^{(1)}} = \delta^{(3)}\frac{dZ^3}{da^{(2)}}\frac{da^{(2)}}{dW^{(1)}} \tag{2.28}$$

To understand $\frac{dZ^3}{da^{(2)}}$, consider Figure 2.16. [33]

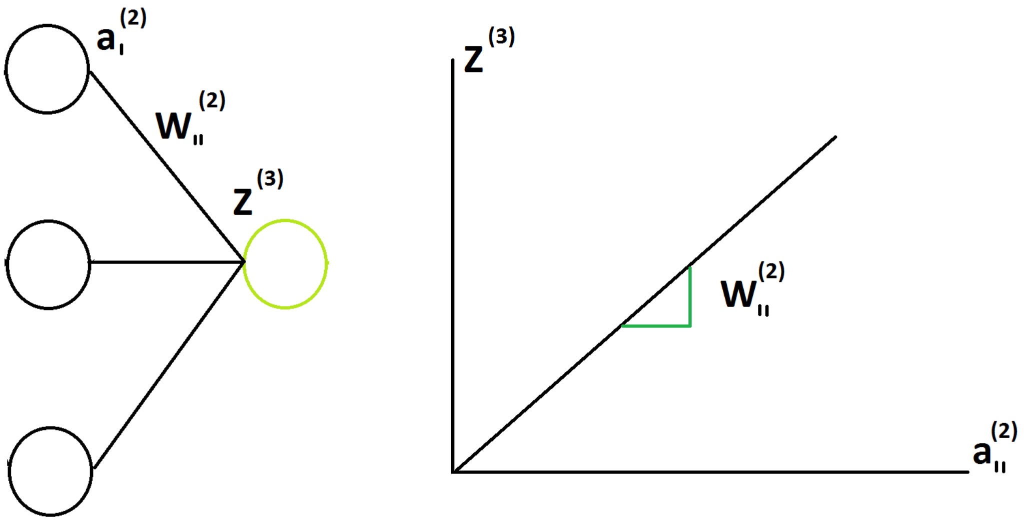


Figure 2.16: *The right part illustrates the relationship between $Z^{(3)}$ and $a_{II}^{(2)}$, where the slope is simply $W_I^{(2)}I$. The left part of this figure illustrates where in the network the variables are located. [33]*

Following the same reasoning as for $\frac{dZ^{(3)}}{dW^{(2)}}$, yet again applying the chain rule, the following can be concluded

$$\frac{dJ}{dW^{(1)}} = \delta^{(3)}(W^{(2)})^T\frac{da^{(2)}}{dW^{(1)}} = \delta^{(3)}(W^{(2)})^T\frac{da^{(2)}}{dZ^{(2)}}\frac{dZ^{(2)}}{dW^{(1)}} \tag{2.29}$$

Using the same reasoning for $a^{(2)} = f(Z^{(2)})$ as was done for $\hat{Y} = f(Z^{(3)})$, it can be concluded that

$$\frac{dJ}{dW^{(1)}} = \delta^{(3)}(W^{(2)})^T f'(Z^{(2)})\frac{dZ^{(2)}}{dW^{(1)}} \tag{2.30}$$

Yet again, following the same reasoning as for $\frac{dZ^{(3)}}{dW^{(2)}}$, but this time for $\frac{dZ^{(2)}}{dW^{(1)}}$, shown in Figure 2.17, it can be concluded that

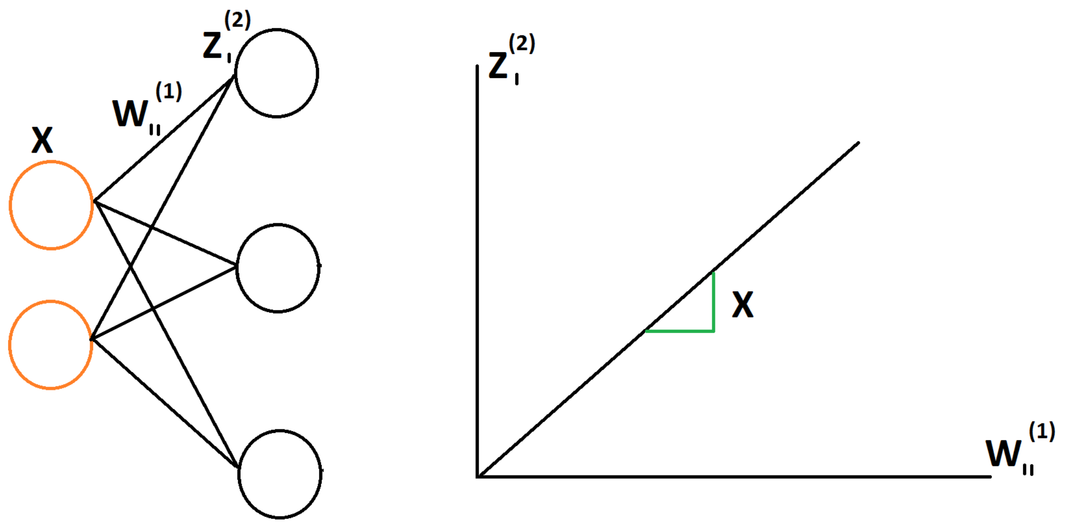


Figure 2.17: *The right part illustrates the relationship between $Z^{(2)}$ and $W_{II}^{(1)}$, where the slope is simply X. The left part of this figure illustrates where in the network the variables are located. [33]*

$$\frac{dJ}{dW^{(1)}} = \delta^{(3)}(W^{(2)})^T f'(Z^{(2)})X^T \tag{2.31}$$

Expressing the backpropagation for the first weights as

$$\delta^{(2)} = \delta^{(3)}(W^{(2)})^T f'(Z^{(2)}) \tag{2.32}$$

allows the expression of the gradient for the first weights as

$$\frac{dJ}{dW^{(1)}} = X^T\delta^{(2)} \tag{2.33}$$

Using equations 2.27 and 2.33, the weights can be updated in the following way:

$$\begin{aligned} W^{(1)} = W^{(1)} - Lr\tfrac{dJ}{dW^{(1)}} \\ W^{(2)} = W^{(2)} - Lr\tfrac{dJ}{dW^{(2)}} \end{aligned} \tag{2.34}$$

where Lr, is the learning rate. [33]

Learning Rate

In order to train the system, the correct learning rate should be chosen. If a too large learning rate is chosen, it can become difficult to find the optimal solution, as shown in Figure 2.18 (a). However, choosing a too small learning rate may also make it difficult to find the optimal solution which is illustrated in Figure 2.18 (b). [27]

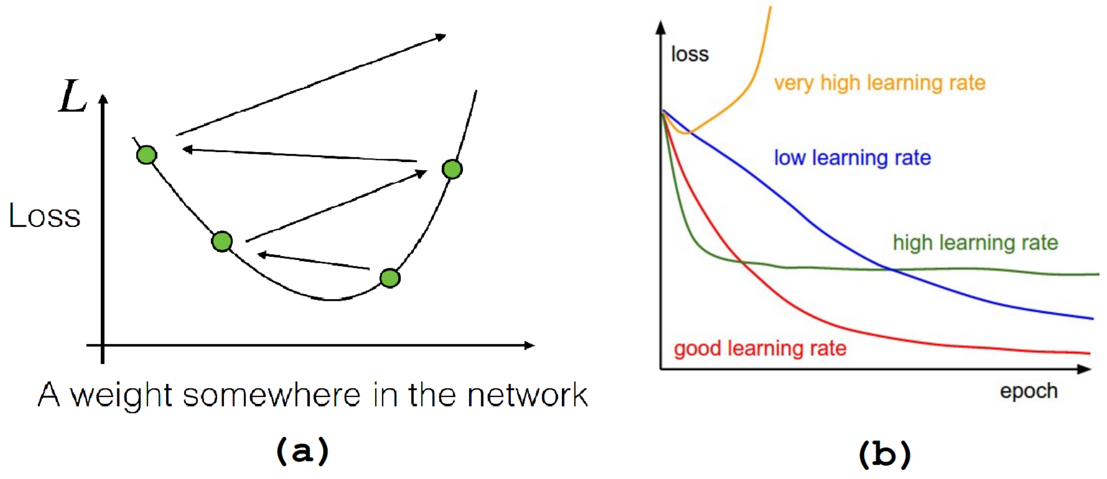


Figure 2.18: *(a) Illustration of what could happen if the learning rate is too large. The green dots indicate values that an arbitrary weight takes and how that corresponds to the loss function of the weight. I.e. having too large learning rate could make it difficult for the network to find the optimal solution, minimizing the loss. (b) Illustration of what the loss curves may look like over several epochs, depending on how the learning rate is chosen. [27]*

Decreasing the learning rate after every epoch is usually helpful. In general, there are three ways of doing this; step decay, exponential decay and 1/t decay. Of the three, step decay is often preferred and simply works by decreasing the learning rate by some factor after a few epochs. [26]

Preprocessing of data

Figure 2.19 illustrates how the data set should be divided. All of the training data is used to train the network and fine tune the parameters of the network while the validation data is used to evaluate the choice of architecture, learning rate etc. After all of the parameters have been chosen and the network trained, the test data is used to evaluate the real world performance of the network. [26]

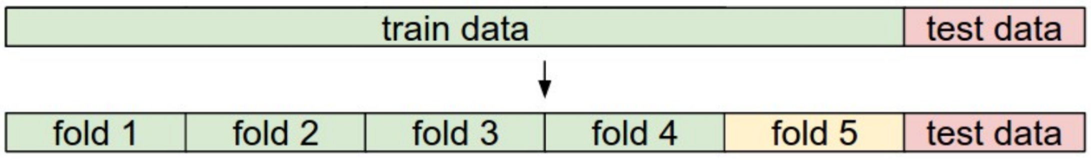


Figure 2.19: *Illustration of how the dataset could be divided. 4/6 should become the training set, 1/6 the validation set and 1/6 the test set. [26]*

Once the data has been split, it is important to normalize it to zero mean and unit variance. This is because normalized data often makes learning problems much better conditioned.
Furthermore, randomly shuffling the order of the data, so that all classes are mixed together, tend to provide a better convergence for the weights in the network [34].

2.5 Kanade-Lucas-Tomasi tracker

The Kanade-Lucas-Tomasi (KLT) tracking method is based on point detection and kernel (which in this context is a patch of an image) tracking. Interest points are important points in images that can be used for tracking. Figure 2.20 shows the interest points extracted from an image using the KLT method (left), in comparison to using SIFT (right), which is a commonly used point detection method. It can be seen that the SIFT extracts more interest points than the KLT and it has been shown that it is generally more robust than most other point detectors. However, more points to track and detect also means higher computational complexity. [12]

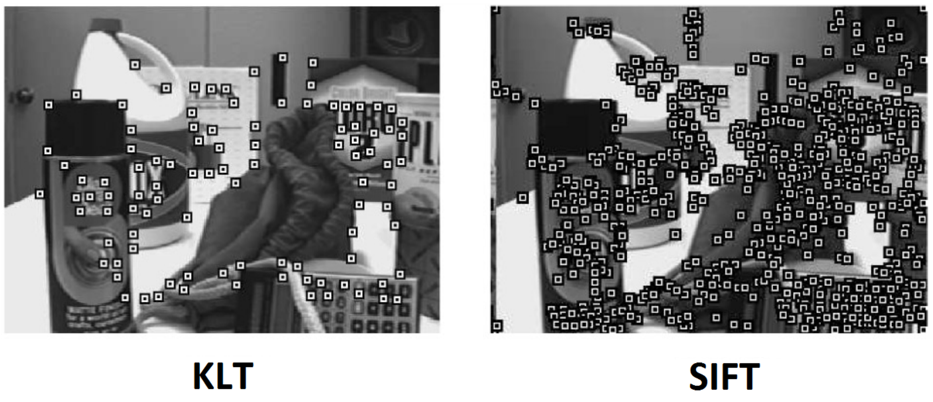


Figure 2.20: *Illustration of the difference between interest points extracted using KLT (left) and SIFT (right) [12].*

In the following parts of this section the KLT method will be described more thoroughly, with focus on interest point extraction and tracking.

2.5.1 Interest point extraction

In order to extract the interest points, the KLT method computes the first order image derivatives in both the x- and y-direction, (I_x, I_y). This highlights the directional intensity variations, which a second moment matrix (M) encodes, and is computed for all pixels in a small neighbourhood. [12]

$$M = \begin{pmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{pmatrix} \tag{2.35}$$

The interest point confidence, R, can then be computed from M by extracting the minimum eigenvalue, λ_{min}, from it. By thresholding R with some value, interest points can be determined. The interest points that have been extracted and lie spatially close to other interest points are removed until only spatially separated interest points are left. Clearly, the KLT method emphasize intensity variations in the image and is both rotation and translation invariant due to the nature of the moment matrix. [12]

2.5.2 Feature tracking

The way that KLT tracks feature is much based on the assumption that, given an image stream, $I(x, y, t)$, where x and y are spatial variables and t is the time of the image, the frame in the image stream a time τ later will be related to the image at time t as $I(x, y, t+\tau) = I(x-\xi, y-\eta, t)$. Meaning it is possible to obtain an image taken at time $t+\tau$ by moving every point in the current image a certain amount. This amount is called the displacement, $\mathbf{d} = (\xi, \eta)$, of the point located at (x, y) between the two time instants t and $t+\tau$. However, in finding this displacement, it is often difficult to only track a single pixel. This is because the pixel can be confused with adjacent pixels and its value can change due to noise. Therefore, KLT tracks windows of pixels instead of single pixels. However, this implies new complications because different points in a window can behave differently since, for example, three-dimensional surfaces can be slanted meaning that its intensity pattern can be warped from on frame to the other. Pixels can also move at different velocities, disappear or appear anew. [35]

Tracking windows creates two problems. The correct window has to be tracked and, when trying to determine the displacement of the window, the different velocities in the window needs to be combined in order to create a single vector. The first problem is solved through residue monitoring, meaning that if the appearance of a window changes too much the tracker discards it. The second problem is solved through modeling changes through an affine map, rather than just simple translations. This way, it is possible to associate different velocities with different points in the window. [35]

Any difference between following windows which a translation cannot describe is considered to be an error. The displacement vector, $\mathbf{d}$, is chosen for the purpose of minimizing this residue error. Let $I(x-\mathbf{d}) = I(x-\xi, y-\eta, t)$ and $J(x) = I(x, y, t+\tau)$ so that the image model is $J(x) = I(x-\mathbf{d}) + n(x)$ where n is the noise between the images. The displacement, $\mathbf{d}$, can then be chosen in order to minimize the following residue error over the window W

$$\epsilon = \int_W [I(x-\mathbf{d}) - J(x)]^2 \omega dx \tag{2.36}$$

where ω is a weighting function. This poses a new problem though, namely how to minimize the residue error. To understand how to solve this problem, consider the intensity function within a small arbitrary window W, like the one illustrated in Figure 2.21. [35]

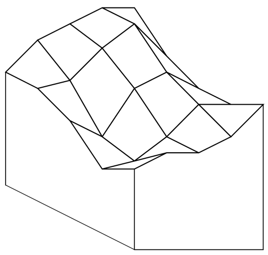


Figure 2.21: *An example of an image intensity function within a small window in an image [35].*

Consider making a copy of it and placing the copy over the first and move the copy horizontally so that a small gap between them is formed. Clearly, the horizontal width of this gap depends on the displacement between the two pieces. When measuring the gap vertically, the width (or height) of the gap is simply the difference between the pieces. For small displacements, both the horizontal and vertical widths of the gap, at a single pixel, are related to each other via the image gradient at that pixel. This means that there are two different ways to describe the volume of the gap in a neighbourhood of a pixel where one way depends on the displacement. [35]

Figure 2.22 illustrates an example of a piece of an intensity function, I(x), and the translated piece behind it. It also illustrates a cross section of the pieces along the direction of the image gradient. [35]

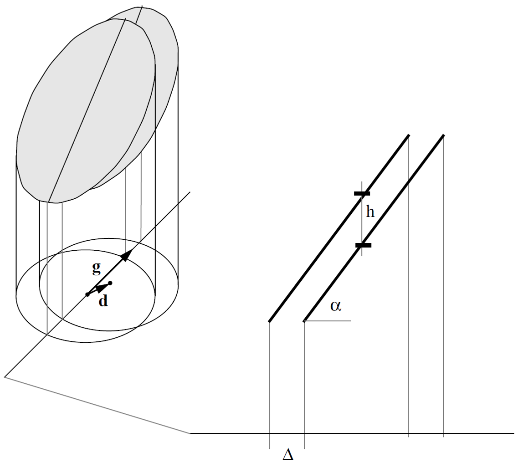


Figure 2.22: *Two image intensity pieces, obtained from a window, from consecutive images (left) and their cross section in the image gradient. Here, Δ represents the projection of the displacement vector,* ***d****, along the image gradient g. [35].*

In general, the displacement vector, **d**, points in a different direction than that which the image gradient, $g = (\frac{\delta I}{\delta x}, \frac{\delta I}{\delta y})$, points. A different way of expressing the image gradient is through its magnitude, κ, and a unit vector **u** meaning that $g = \kappa\mathbf{u}$. The displacement measured along the gradient direction, Δ, can also be expressed with the unit vector, **u**, by projecting **d** along **u** so that $\Delta = \mathbf{du}$. Also, from the right hand side of Figure 2.22 it can be seen that the vertical width of the gap, $h = I - J$, where J is the lower of the two intensity functions, can be expressed as $h = \Delta \tan \alpha$ where α is the maximum slope in the window. It follows that the tangent of α is equivalent to the magnitude of the gradient, κ, meaning that we can write

$$h = \Delta\kappa = \mathbf{du}\kappa = \mathbf{d}g \tag{2.37}$$

This is an equation relating the difference between image intensities, h, the inter frame displacement, $\mathbf{d}$, and the image gradient g. The difference, h, can easily be computed from both images while the image gradient only needs one image to be estimated. However, due to a complication known as the aperture problem, illustrated in Figure 2.23, it is not possible to determine the displacement, $\mathbf{d}$, by just looking at a small piece of the image edge (like the pieces illustrated in Figure 2.22). Rather, it is only possible to determine one component of $\mathbf{d}$ [35]. The aperture problem states that if the motion of an edge is to be estimated by studying a small piece of it, which is small in comparison to the entire edge, the only motion that can be estimated is the one that is perpendicular to the local orientation of the edge, as illustrated in Figure 2.23. [35], [36]

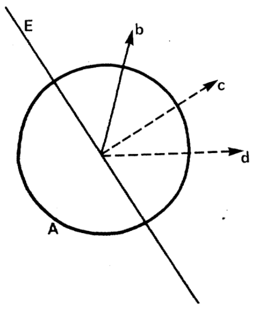


Figure 2.23: *Illustration of the aperture problem. The area, A, of the edge, E, that is studied is small which makes it impossible to determine the motion of the edge in any other direction than c. [36].*

However, the window W, is larger than the small pieces of it that have been discussed above. This might allow the computation of different components (assumed to be constant within W) of the displacement vector inside different pieces of W. When combining all of the estimated components it is possible to determine if an assignment is wrong if the right hand side of equation 2.37 is not the same as the left hand side. Choosing a displacement vector, $\mathbf{d}$, which minimizes the square of that difference, integrated over the entire window, is the best choice possible, i.e. minimizing the weighted residue

$$\int_W (h - g\mathbf{d})^2 \omega dA \tag{2.38}$$

with respect to $\mathbf{d}$ will yield the best $\mathbf{d}$. However, the solution $\mathbf{d}$ to equation 2.37 will often contain some error due to it only being true when $\mathbf{d}$ approaches zero or when the image intensities are linear functions of the coordinates x and y. Although, an

approximation of **d** can still be computed and the basic step 2.37 can be repeated on an image which have been resampled through bilinear interpolation, which is a method making it possible to realistically scale up images to achieve sub-pixel accuracy. After a few such iterations, **d** will converge. [35]
A negative property of this method is that different parts of an image contributes with different success for tracking. For example, if the intensity pattern, I in a part of the image is constant, the matrix of gradients, **G**, will be null and the displacement, **d**, will be impossible to compute. [35]

2.6 Additional theory

This section presents some additional theory used in this thesis.

2.6.1 Histogram of oriented gradients

The basic idea of histogram of oriented gradients (HOG) is that the distribution of local intensity gradients, i.e. direction of edges, often represent object shape and appearance quite well. Obtaining the HOG is in practice done by dividing the image into small regions called cells (often with a size of 8x8 pixels). In these regions the local one-dimensional histograms of gradient directions over the pixels in the cell is accumulated. In order to be more persistent to noise and illumination, the local histograms over larger regions called blocks (usually with a size of 4x4 cells) are obtained and the result is used to normalize the cells in the block. The normalized histograms of the blocks are then concatenated in order to get a feature vector describing the image. This feature vector can in turn be used for classification. Figure 2.24 shows a simple schematic of how the HOG of an image is obtained. [37]

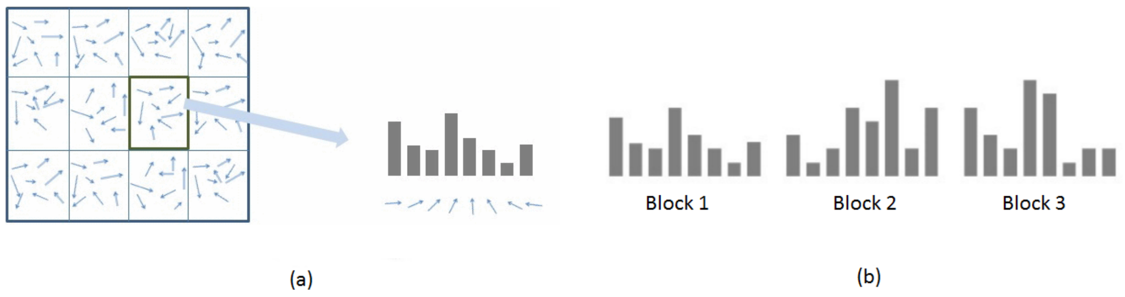


Figure 2.24: *(a) HOG obtained from cells, the figure does not show the blocks used for normalization. (b) Histograms of the blocks concatenated into a feature vector. [38]*

2.6.2 Viola-Jones algorithm

The Viola-Jones algorithm is an object detection framework able to detect sought objects in real time. The incentive for the algorithm was primarily to detect faces in images (hence this section will be about face detection) but it can be used for other objects as well. The algorithm consists of four stages which will be described below. [39]

Haar feature selection

The features used for training and detection makes use of the fact that human faces share some common properties. Some of these properties are that the eye region is darker than the upper-cheeks and that the nose bridge region is brighter than the eye region. The Haar features rely on rectangular areas as those seen in Figure 2.25 and the value of a given feature is the sum of the pixels within the shaded rectangles minus the sum of the pixels inside the white rectangles. These features are obtained from training images and are sought after in following images.

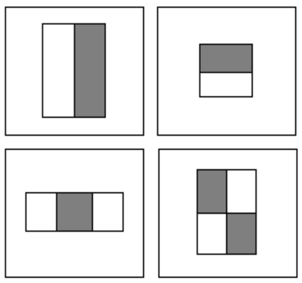


Figure 2.25: *Haar features used in the Viola-Jones algorithm [39].*

Creating an integral image

An integral image evaluates the Haar features and computes them very rapidly. The integral image at position (x, y) contains the sum of the pixels above and to the left of this position and is calculated as

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x', y') \tag{2.39}$$

in which ii is the integral image and i is the original image. By using the integral image, one can calculate any rectangular sum with the use of four array references, see Figure 2.26. By doing so the difference between two rectangular sums, such as the rectangles in Haar features, can be computed with eight (or six if the rectangles are adjacent) references. Using these references, the evaluation speed of the Haar features is increased. [39]

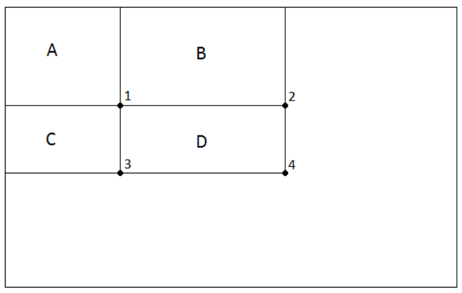


Figure 2.26: *The sum of the pixels within rectangle D can be computed with the four array references. The value, in the integral image, at location 1 is the sum of the pixels in rectangle A, the value at location 2 is A + B, the value at location 3 is A + C and the value at location 4 is A + B + C + D. With these values the sum within D is computed as 4 + 1 - (2 + 3). [39]*

Adaboost training

To reduce training time and also increase predictive power, the AdaBoost (Adaptive Boosting) is used. This training process selects only the features that are known to improve the predictive power of the trained model and improves the execution time as irrelevant features do not need to be computed. [39]

Cascading classifiers

To classify an image, a cascade of classifiers is used. This means that several binary complex classifiers are put together and the candidate image passes through these classifiers one at a time. The complexity of the classifiers increase as the image proceeds through the classifiers. If in any of the classifiers the image is rejected, the cascade is terminated. By terminating, time is saved by not invoking the computation-intensive classifiers further down the cascade. Using a cascade of classifiers increase both the detection performance as well as reduces the computation time. [39]

2.6.3 Kinect tracking algorithm

The Kinect tracking algorithm is presented here so that the reader understands the basics of the algorithm which the algorithm developed in this thesis is compared to.

The Kinect consists of an IR laser projector, an IR camera and a RGB camera. Furthermore, the tracking algorithm used in the Kinect consists of primarily two stages. The first stage involves computing a depth map by using structured light. Structured light means that a depth map is constructed by analyzing a known speckle pattern, seen in figure 2.27, of infrared laser light that is present on the wanted object. This known pattern is projected onto the scene by the IR laser projector and the pattern, which will be deformed due to different depths in the scene, is then read by the IR camera. From this reading and deformation a depth map can be created. To increase the precision of the depth map, the Kinect uses depth from focus and depth from stereo. Depth from focus uses the principle that objects appears more blurry if they are further away. Depth from stereo uses the principle that by looking at the scene from another angle, objects being close to the camera will be more shifted to the side than objects far away from the camera. [40]

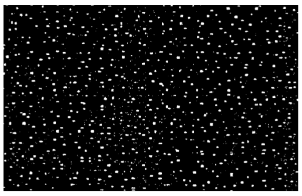


Figure 2.27: *Known speckle pattern that is projected onto the scene [40].*

The second stage involves machine learning, i.e. teaching the Kinect how to interpret the information obtained from the depth map. In this stage around 100 000 depth images with known skeleton models from a motion capture system are used. From each of these depth images dozens more are rendered in order to represent a lot of different body types. Based on these images the algorithm learns a randomized decision forest in order to be able to classify the skeleton model from a specific depth image. When the depth map has been classified by the randomized decision forest the algorithm creates a skeleton representation using mean shift. [40]

With this algorithm the Kinect is able to, at a distance of 1.6 meters, determine an objects position with an error range of 1-2 centimeters at 31 FPS [40], [41].

2.7 Related work

Many different articles and approaches were reviewed in order to develop the algorithm in this thesis. Here, two selected articles will be mentioned.

In the paper [42] the authors describe how they developed a marker-less hand/finger tracker to enable human-computer interaction. For their solution they use the YCbCr space for both background subtraction and skin extraction. The Viola-Jones algorithm is then used to detect and remove the face and segmentation based on Canny edge detection is performed to distinguish the hand from head if the head is behind the hand. As programming language they chose *C#*. This solution has overcome the limitations of long sleeves, background without skin-colored objects and hand and head not overlapping. The tracking of their algorithm is robust, but it has some limitations of its own. For example, according to the authors, it requires users to stand 40-100 cm from the camera and that the environment has sufficient lightning conditions. [42]

In the article [43] an algorithm for tracking the face and arms of a human is described. The algorithm is initiated by standing in front of the camera in a certain pose. From this pose information such as arm length and shoulder width is extracted. During the run time of the algorithm, hypotheses (based on the information from the initialization) are generated in order to estimate the position of the arms and head. Later on, these hypothesises are verified with the use of color and edge information and the best hypothesis is assumed to be the correct position of the arms and head. According to the article the tracking works in cluttered environments and it does also overcome the limitation of long sleeves. However, since it needs to be initialized with a certain pose, it might be difficult to use in rehabilitation since certain patients might not be able to achieve this pose. [43]

3 Methods

Following are the different software, hardware used as well as a description of how the algorithm was implemented. A short summary of rejected approaches is also presented.

3.1 Software

The software used for developing the code in this thesis is MATLAB (R2015b, MathWorks, Natick, USA). This is due to the fact that MATLAB has built in support for working with images and is a good tool for implementing and evaluating algorithms. [44]

3.2 Hardware

The laptop that was used for performance evaluation of the algorithm is a Lenovo G50-80, equipped with an i5-5200U dual core processor at 2.20 GHz and 6.0 GB RAM. This laptop is equipped with an internal web camera capable of using the formats RGB24 and YUV with a maximum resolution of 720x480 pixels. [45]

The integrated web camera in the laptop turned out to have a permanent auto focus feature which caused major complications during the development of a reliable background subtraction. This since auto focus changes pixel values in the image resulting in a non-static background. Since this thesis is limited to dealing with a static background, see Section 1.2, another camera without auto focus was used instead. This camera is the external web camera Logitech HD Webcam C270 which has a maximum resolution of 1280x720 pixels [46]. The resolution used was 640x480 pixels with the format RGB24. This since such an amount of pixels proved sufficient and dealing with more pixels would increase computation time.

3.3 Implementation of algorithm

In the following sections the major parts of the algorithm and their implementation are described. The parts are written in their order of implementation. The theory behind the different methods implemented is described in chapter 2. Furthermore, in the end of this section, a flowchart of the entire algorithm is presented.

3.3.1 Object detection

First of, the object, i.e. the human, needs to be detected. It is only when the human has been detected that its hands and head can be tracked. The detection is done with the two major parts described below.

3.3.1.1 Background subtraction

Two different approaches for background subtraction were examined, adaptive and non-adaptive background subtraction. Adaptive background subtraction proved to give decent results. However, since adaptive background subtraction creates a model of the current background, if the object to be detected is temporarily static (or has slow movements), it might shift from the foreground to the background resulting in a non-detectable object. Since the intended users of this algorithm are meant to be patients with stroke or COPD who, most probably, are slow in their movements there is thus a risk for the patients to become a part of the background. Such an event will be difficult to handle since the patient will be a non-detectable object.

Therefore, a non-adaptive approach was selected. The main disadvantage of such an approach is the fact that there is a need to, in advance, create a background model with the user excluded from the scene. This inconvenience will result in a slightly less user-friendly algorithm. However, the advantages of such an approach is its simple implementation and the fact that the user can perform slow movements as well as be static while still being a part of the foreground.

To get a reliable background model, 31 images of the background are captured. The median of each pixel location throughout these images is calculated and these median values will form the background model. By using several images instead of a single image to model the background, random fluctuations in pixel values will have small or no impact on the background model. Furthermore, by using median the background model is not affected by a few abnormal values, as in opposite to the mean. When the background model has been created it is used together with thresholding on future frames in order to obtain the foreground. Thresholding in both the RGB and YCbCr color space were examined.

In order to further improve the performance of the background subtraction some additional methods are used. First of all, morphology is used to smooth edges and reduce noise. To counteract noise even further, a median filter is used along with a function that disregards objects with a size less than a predetermined amount of pixels as well as a function that fills holes in detected objects. Shadow removal was also tested in this stage. Therefore it was not included in the final algorithm. By using the aforementioned methods a silhouette, seen in Figure 3.1, will be obtained. This silhouette can then be filled with the original image and thus only the sought object is obtained, see Figure 3.1. The remaining parts of the algorithm can thereafter work with solely this image.

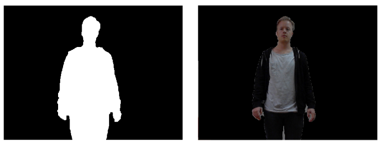


Figure 3.1: *Silhoutte obtained by background subtraction, to the left. Silhoutte filled with original image, to the right.*

3.3.1.2 Skin extraction

In theory, the background subtraction will be able to extract only the human from a scene. In order to extract only the hands and head, skin extraction is done, meaning that only skin colored objects are extracted from the scene. Provided that the only skin visible on the user is the head and the hands, i.e. the user is wearing long sleeves and trousers, and that there are no clothes with colors similar to skin, the skin extraction should, in theory, result in that everything except the hands and head is disregarded (meaning that these pixels are set to the color black in our algorithm). If this is the case, then there will, ideally, be three separated areas (given non-occlusion and that no body part is in contact with another) that can be sent to the classification step of the algorithm.

Thresholding in three different color spaces (RGB, HSV and YCbCr) were examined. Combinations of these color spaces as well as only the individual color spaces themselves were tried and different thresholding in each individual channel was tested. The values used for thresholding were found by examining skin-colored objects in each color space.

In order for the algorithm to adapt to different statical conditions, such as skin color, the skin extraction is initialized by finding the head of the user with the Viola-Jones algorithm. When the head has been found, eleven images of the head are captured. In these images, the median value of each color channel in the RGB and HSV color space is calculated. From the eleven medians, a mean for each color channel is calculated. Doing so will result in stable and correct values, regardless of the statical conditions, for the mean values of each color channel. When these values are obtained, skin extraction is done by thresholding around these values.

To set things up for the classification step, the areas obtained from the skin extraction, referred to as bounding boxes, are all set to the same size. If less than three bounding boxes are found, meaning most probably that one body part is occluded, the skin extraction is iterated until three or more areas are found and before this the classification step is not initialized. Furthermore, to decrease the amount of bounding boxes and therefore decrease the time taken by the classification step, bounding boxes that are close to each other will be merged into a common bounding box at a position determined by the bounding boxes being merged. This step is also helpful as sometimes, due to lightning conditions, a body part might be detected as two separate areas. The images in the bounding boxes are also converted to gray scale before being sent to the classification step.

So to conclude, the input in the skin extraction step is an image ideally containing only the users body and the output is a number of bounding boxes (three or more with a size of 70x80 pixels) containing gray scale images with skin colored objects that are candidates for being either a hand or a head.

3.3.2 Head and hand classification

When the bounding boxes have been obtained they are sent to the classification step of the algorithm. The purpose of this step is to determine which boxes that contains the hands and head. This is done by giving each box a score representing the probability that it is a hand or a head. Two major approaches for classification were examined, SVM and CNN. The reason for this is that the authors were unsatisfied with the performance of the SVM thus CNN was examined. However, it was unknown if CNN would be able to perform in real time and therefore the SVM was developed. It turned out that the CNN had similar computation time as the SVM hence, in the final algorithm, CNN is used for classification.

3.3.2.1 SVM

Training a SVM can be done in many different ways. The factor with the biggest impact is the choice of features. Other parameters with great influence on the training process are the choice of kernel and the amount of training data.

Several different features and combinations of features were examined. Amongst the involved where HOG, LBP, different color spaces and gray scale histograms. Different kernels were also examined and amongst the tested were linear, Gaussian and polynomial. Furthermore, training images in different quantities were obtained from the original images, the background subtracted images and the skin extracted images. The quantities ranged from tens of images to hundreds of images.

Using HOG as feature vector (i.e. calculating the HOG from the images resulting in a feature vector in $\mathbb{R}^{1260}$) along with a linear kernel as well as 400 training images (100 on each hand and 200 on the head) obtained from the skin extraction gave the best results. The images were obtained by letting one of the authors pose at a distance of about 2 m from the camera. To reflect the many possible looks of a hand, a large variety of hand expressions, such as fist, claw and palm, were captured in the training images for the hands. The images were captured on the side, front and back of the hands at different positions around the body and was taken with a resolution of 70x80. For the training images of the head, different facial expressions as well as different tilting and rotation that always showed both eyes of the head were captured. By using training images from the skin extraction, interferences in the training process from clothes and background are avoided.

After training the SVM with the parameters and training images aforementioned the trained model can predict if a new image (with the same size as the training images) is either a hand or a head by calculating the posterior probabilities (ranging from 0 to 1) that a new image belongs to a certain class. In order to determine the performance of the SVM, the misclassification rate was calculated.

3.3.2.2 CNN

A major advantage with using CNN is that it does not require the extraction of a certain feature to work properly since it basically creates its own unique feature extraction method that best fits its purpose. There are however, as with the SVM, a lot of parameters to be chosen. These are the structure of the network, number of filters and filter depths, the type of striding and padding for each corresponding block, the pooling filter size, the type of activation function, the dropout rate, the type of loss layer and finally the structure and appearance of the data used to optimize the network.

In this thesis, a network that was not only reliable, but also fast, was needed. This meant that the most basic structures of CNN should be used (such as the one illustrated in Figure 3.2). In the resulting CNN, the input is as aforementioned a grayscale image of the dimensions 70x80. Furthermore a convolutional layer with nine filters with filter size 5x5 is used to create nine different feature maps. These feature maps are then scaled down by a factor of two through a maxpool layer and are then processed by a FC layer. This is done so that the output layer easily can be created through another FC layer, which has an output size of 2x1x1, i.e. two numbers. Through the use of a softmax layer, these numbers are turned into probabilities corresponding to one of the two classes, head and hand. Also, after the convolutional layer and both FC layers, there are ReLu layers which are not illustrated in the figure. After the maxpool and second ReLu layer there are also dropout layers with a dropout rate of 0.5. Furthermore, the input data has been preprocessed, to ensure that the CNN is optimized by changing the data type to single and subtracting the image mean.

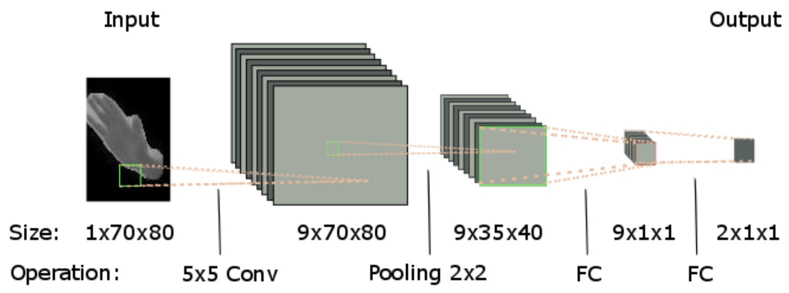


Figure 3.2: *Illustration of a basic structure of a CNN used in this thesis. What is not illustrated in this image is the ReLu, dropout and loss layers. The ReLu layers are situated directly after each convolutional and FC layer while the dropout layers (with a dropout rate of 0.5) are placed strictly before each of the FC layers. Finally, the loss layer is placed last and is of the softmax type.*

After the CNN had been created, another major task presented itself, namely how to train the network in the most effective way. Since a CNN is trained better the more training data it has, 1786 unique grayscale images of hands and heads were used, where the skin had been extracted. The images were evenly distributed between the two categories and all of the images were acquired by letting the authors pose for pictures during a constant and sufficient lightning condition at a distance of about 2m from the camera. A resolution of 70x80 were used for these images. The hand images were taken evenly on left and right hand as they moved around the body and made different poses such as fist, claw or palm. These images were taken on the front, side and back of the hands. Furthermore, the head images were taken as the authors made different poses and looked to the sides, but always showing both eyes. Of the images, 4/6 were used as training data, 1/6 as validation data and 1/6 as test data. Before training, the order of all of the training and validation images was randomized to reduce overfitting. The training data was then distributed in several batches that was used for training. A lot of time was put on fine tuning the learning rate and batch size to achieve the smallest possible misclassification rate and an optimal loss rate. To estimate the final misclassification rate after training, the test data was used. Similar to the SVM, posterior probabilities are obtained in order to classify the image.

Figure 3.3 illustrates a more complex structure than the one illustrated in Figure 3.2. This structure was used to evaluate the benefits and drawbacks of having a deeper network in the algorithm.

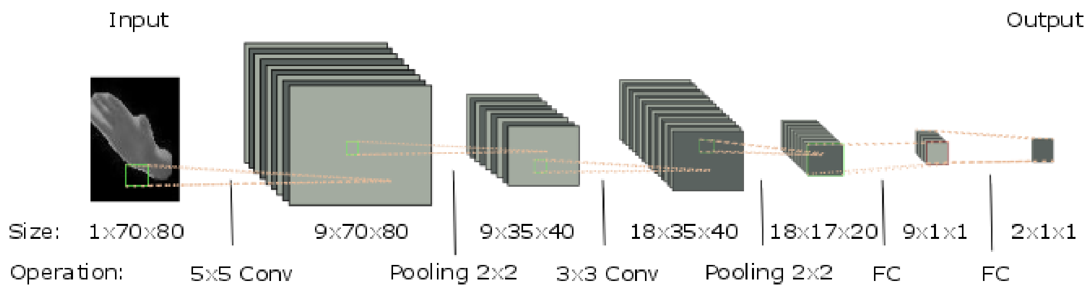


Figure 3.3: *Illustration of the more deeply structured CNN used in this thesis. Not illustrated in the image are ReLu layers which are present directly after each of the convolutional and FC layers. There are also dropout layers with a drop out rate of 0.5 before each of the FC layers. Finally, softmax is used as the final loss layer in this structure as well.*

3.3.3 Object tracking

In order to track the object, once it has been detected and classified, KLT is used. When the location of one of the sought body parts has been found, KLT is used to extract interest points in that region. However, since KLT is prominent to finding interest points on edges, the raw image data is used rather than the one that has been processed through background subtraction and skin extraction. The raw image data is used in its gray scale representation to make the interest point acquisition faster. Since the raw image is used, it is possible that some interest points are acquired in the background. For example, if the hand is over a striped shirt then some interest points may appear on these stripes, which of course is unwanted. To solve this, the foreground and skin extracted version of the image is used to determine if some interest points lie in the background, in which case they are removed.

Once the qualified interest points have been determined, they are used to initiate the tracker. This tracker is then used to track the points in the image using the recently acquired raw gray scale image. However, since it is possible that the tracker does not perform a reliable tracking, background subtraction and skin extraction is performed to determine if any of the points have been tracked into the background, in which case they are removed and the tracker is reinitialized with the remaining points. Another problem that the KLT can run into is that many of the interest points are lost if the tracked object changes too much in appearance, such as if an open hand turns into a fist. In order to solve this issue, new interest points are extracted in the foreground every fifth frame. These new points are merged with the previously tracked points and together they reinitialize the tracker. If all the tracked interest points are lost or become too few for a reliable tracking, object detection and classification is performed again until all three body parts have been detected and classified. Once this happens, interest points are extracted from the new areas which are used to reinitialize the tracker and the tracking is resumed.

Since three body parts are tracked separately, the tracking and object detection as well as classification is performed in three different parallel pools in MATLAB. These parallel pools allows the computers processing power to be divided evenly between the three tasks and perform them simultaneously. This may however decrease the overall processing time.

3.3.4 Entire algorithm

A flowchart of the entire algorithm can be seen in Figure 3.4. As can be seen, the algorithm is initiated by obtaining a background model and the main loop, marked with a red dashed line, is entered when the hands and the head has been correctly classified and when the tracker has been initialized with good interest points. The main loop is iterated once for each frame. It starts by detecting skin colored areas. From these areas, points are extracted and tracked. After this, a new iteration takes place.

There are some diversions available from the main loop. At predetermined time intervals, to ensure that the correct objects are being tracked, objects are to be detected and classified again. If no hands or head are found at this stage this step is iterated until they are found and in the meantime they are tracked at their last known location. Not seen in the flowchart is that this continuation of tracking goes through the normal process of removing and adding new interest points if needed. The tracker is reinitialized with the points found in the new areas and a new iteration takes place. Furthermore, in the main loop, if the number of interest points are less than a predetermined number, more interest points are extracted and the tracker is reinitialized. This to ensure a sufficient amount of points available for tracking.

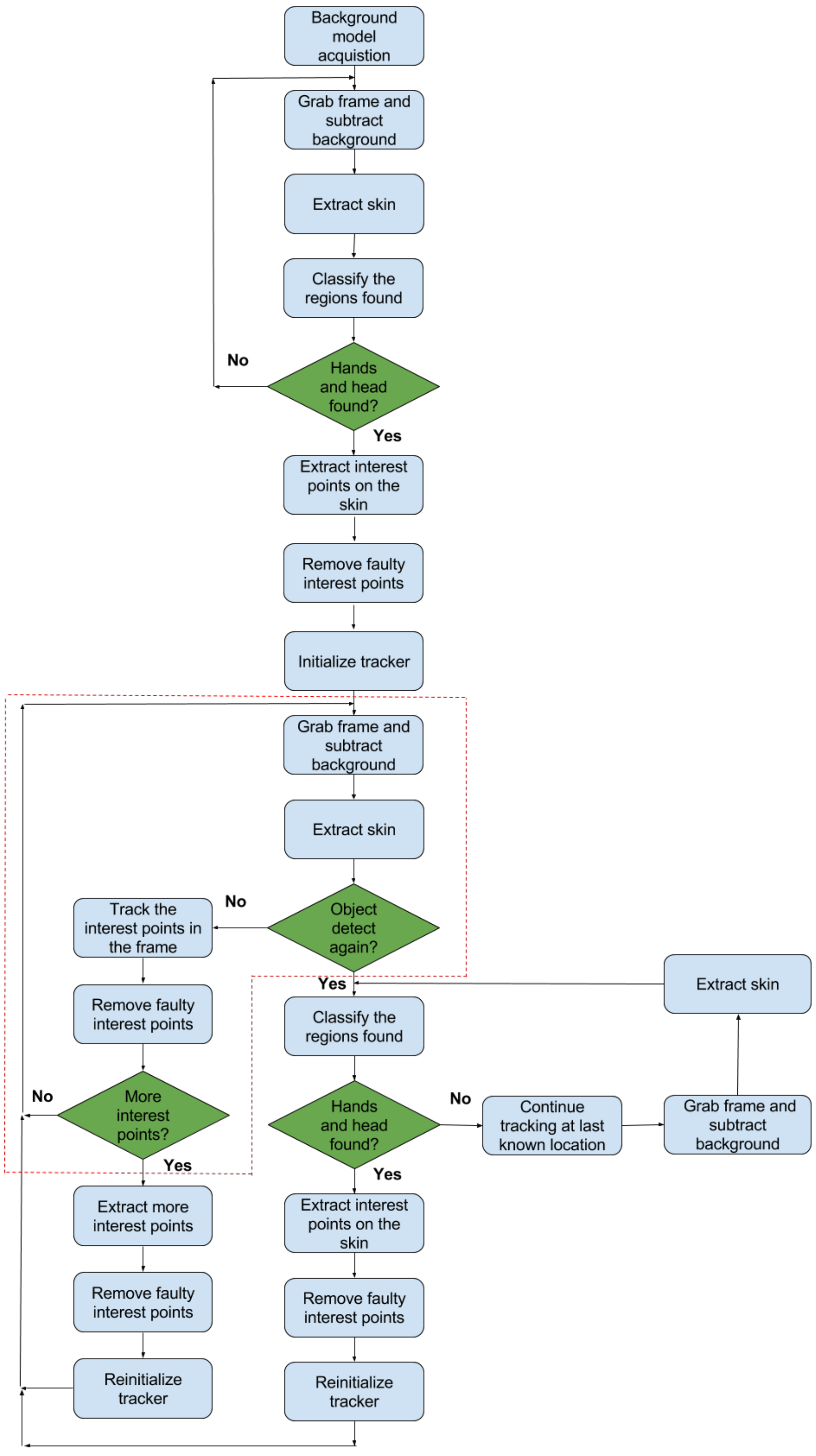

Background model acquistion
Grab frame and subtract background
Extract skin
Classify the regions found
No
Hands and head found?
Yes
Extract interest points on the skin
Remove faulty interest points
Initialize tracker
Grab frame and subtract background
Extract skin
No
Object detect again?
Track the interest points in the frame
Yes
Extract skin
Classify the regions found
Remove faulty interest points
No
More interest points?
Hands and head found?
No
Continue tracking at last known location
Grab frame and subtract background
Yes
Yes
Extract more interest points
Extract interest points on the skin
Remove faulty interest points
Remove faulty interest points
Reinitialize tracker
Reinitialize tracker

Figure 3.4: *Flowchart of the proposed algorithm. Main loop marked with a red dashed line.*

3.4 Rejected approaches

The original approach was to use non-adaptive background subtraction for detection. The user, once detected, would initialize the tracking by placing his or hers hands and head in three separate boxes, i.e. kernels. Different features would be extracted from these kernels and in the next image frame a uniform search around each kernel's previous position would be done in order to find the new position of the corresponding kernel. In this uniform search, features from the previous kernel were to be compared, in different ways, to features in the new kernels in order to find the best match. To avoid noise and to allow the sought objects to change shape, the idea was to update each kernel iteratively as the frames moved on. So the kernel template would consist of, for example, the ten previous kernels weighted in a certain order so that the most previous kernels would have the greatest impact.

Initially, the features extracted from the kernel was the color histogram and the LBP. The idea was that by using color and LBP, i.e. texture, one would be able to distinguish the hand from the head and vice versa as well as both the hands and the head from other objects. However, this proved to be difficult and other features such as GLCM, HOG, edges, different point detectors and combinations of these were tested as well. None of these gave satisfactory results. The features were also compared in different ways, for example by using euclidean distance and the Bhattacharyya coefficient. Furthermore, optical flow was tested in order to predict in which direction to search in the next frame.

4
Results

Since each part of the developed algorithm is, to a large extent, individual it can be evaluated and discussed on its own. Therefore, the results, as well as the discussion, of this thesis will be divided into sections representing each part of the algorithm. To wrap up, the performance of the algorithm in whole will be presented. The results presented have been obtained by letting only the authors be the users of the algorithm. Furthermore, the algorithm was only tested in one specific room.

4.1 Object detection

This section will cover the results from the detection step of the algorithm, i.e. the background subtraction and the skin extraction.

4.1.1 Background subtraction

In Figures 4.1 - 4.4 the result of the background subtraction is seen. The background subtraction used in these results is explained in Section 3.3.1.1 and the YCbCr color space is used for thresholding[1]. Figures 4.1 - 4.3 show the outcome of the background subtraction when the background model, seen as the leftmost image in each figure, is increased in difficulty. As can be seen and as expected, the background subtractions performs worse, i.e. the amount of visible non-human objects increase, when the background increases in difficulty.

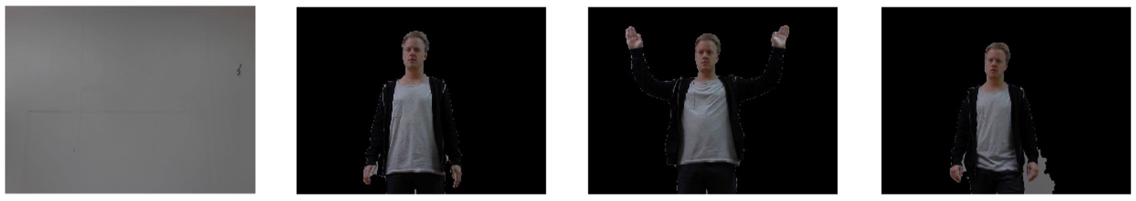


Figure 4.1: *Background subtraction with a background of easy difficulty.*

[1]Threshold levels found in appendix B.1

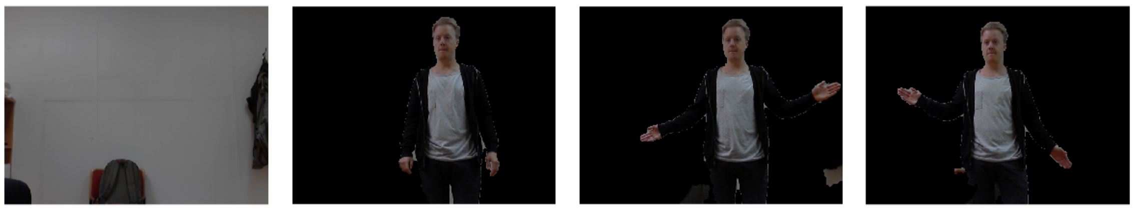


Figure 4.2: *Background subtraction with a background of medium difficulty.*

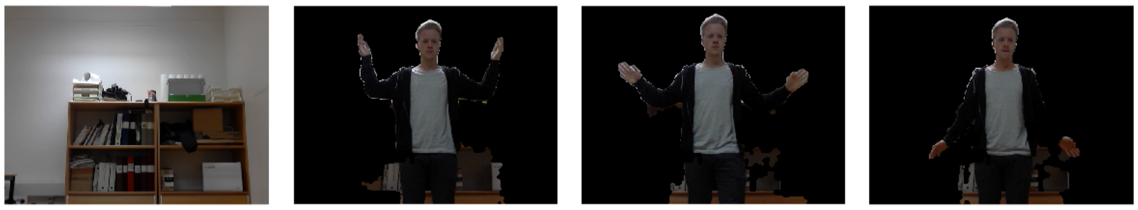


Figure 4.3: *Background subtraction with a background of hard difficulty.*

In figure 4.4 there is no color difference between the t-shirt and the background (as in figures 4.1 - 4.3 where a long sleeved shirt forms a barrier) resulting in that parts of the t-shirt are considered to belong to the background due to its similar color. However, as long as the background is not skin colored this will not be a problem.

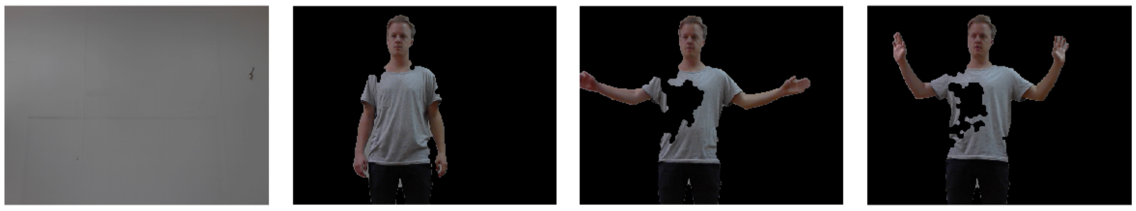
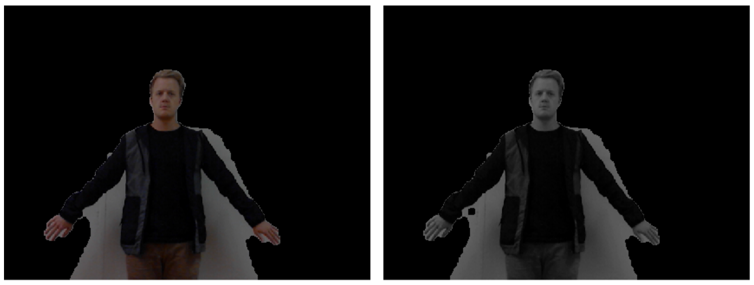
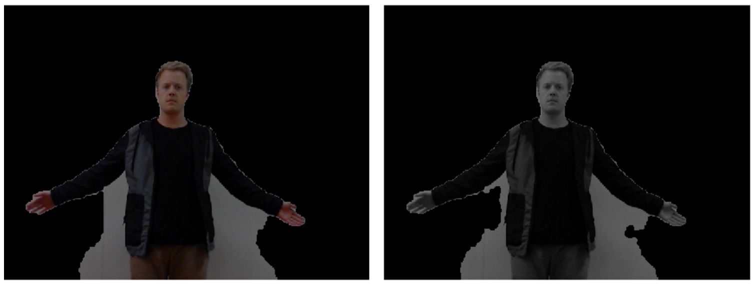


Figure 4.4: *Background subtraction with a background of easy difficulty in which a piece of clothing is sometimes considered to belong to the background.*

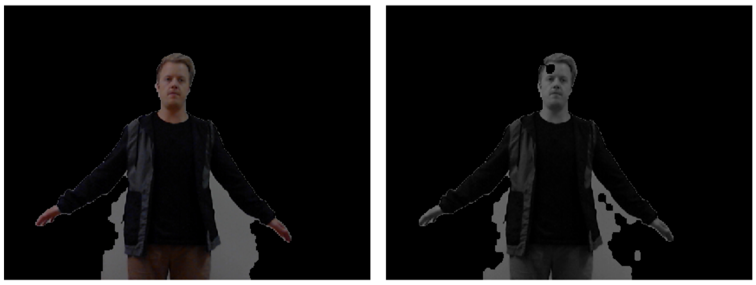
The above images have been obtained without the implemented shadow removal. As can be seen in the rightmost image of Figure 4.1, a bit of the background is visible due to shadows. Furthermore, the less homogeneous background the more shadows, meaning that more background is present in the image. In Figure 4.5 the result from the developed, but not implemented, shadow removal is shown. The shadow removal was not implemented since it was concluded that it did not work good enough and under certain conditions it worsened more than it helped. As can be seen, standing closer to the wall will obviously result in more shadows and in such a scenario the shadow removal removes a very small amount of shadows. When moving from the wall, the shadow removal is gradually increased in performance. When standing 30 cm from the wall, a small part of the head is considered to be shadows and hence removed. If large portions of a head or hand are removed this way the rest of the algorithm might become compromised.



(a)



(b)



(c)

Figure 4.5: *Background subtracted image to the left. Shadow removal to the right. (a) Standing 10 cm from the wall (b) Standing 20 cm from the wall (c) Standing 30 cm from the wall.*

4.1.2 Skin extraction

In Figure 4.6 and 4.7 the result of the skin extraction (applied on images obtained from the background subtraction) can be seen for two different backgrounds. The skin extraction used in these results is explained in Section 3.3.1.2 and a combination of the RGB and HSV color spaces is used for thresholding[2]. Due to the fact that parts of the background in Figure 4.7 has a color similar to that of skin, the skin extractions performance is impaired.

[2]Threshold levels found in appendix B.2

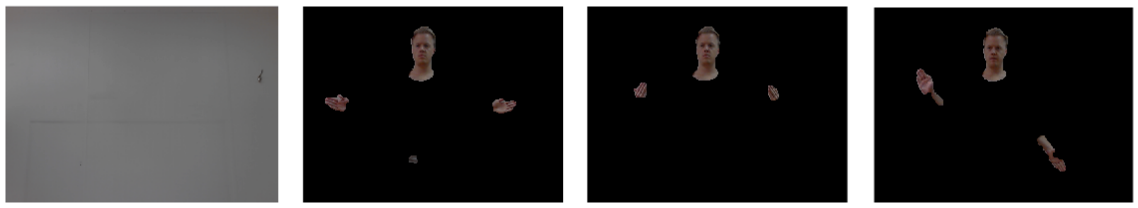


Figure 4.6: *Skin extraction with a background of easy difficulty.*

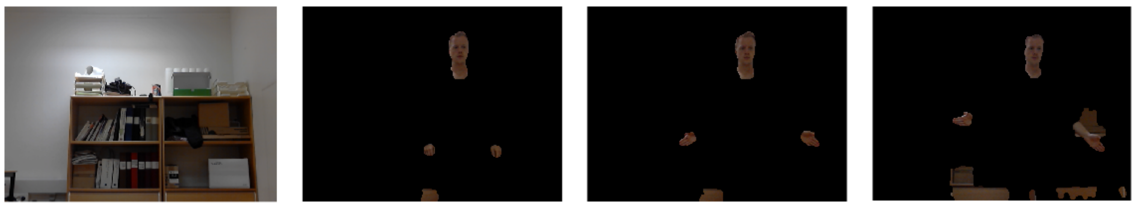


Figure 4.7: *Skin extraction with a background of hard difficulty.*

Figure 4.8 shows a number of objects with different colors. There are in total eight different objects, in rows of four, and the top images show the object before skin extraction while the bottom images shows the object after skin extraction. As can be seen, certain colors are more difficult to occlude, such as yellow and brown.

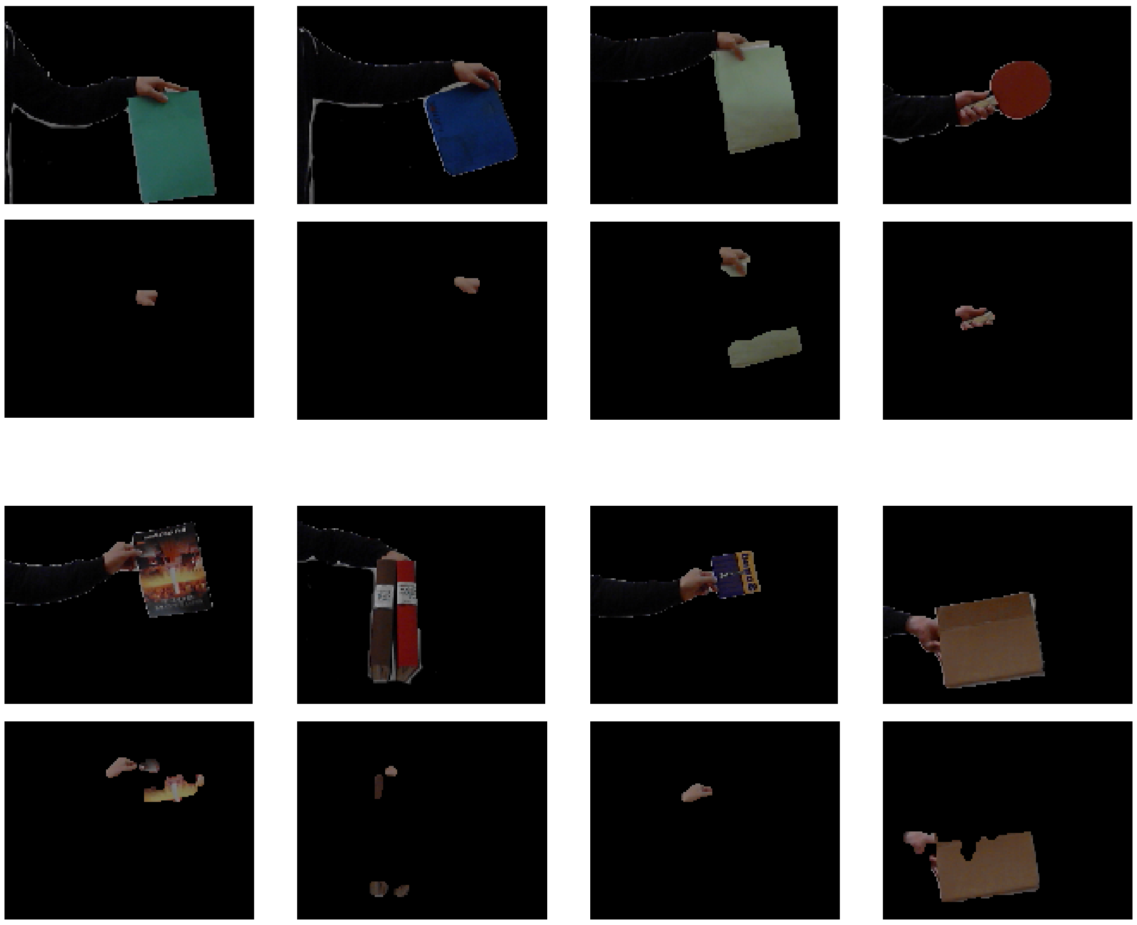


Figure 4.8: *Skin extraction on objects of different colors.*

When the skin has been successfully extracted, as in Figure 4.9, bounding boxes are extracted, transformed to gray scale and sent to classification. The bounding boxes for the head is not covering the entire head since all bounding boxes are set to the same size.

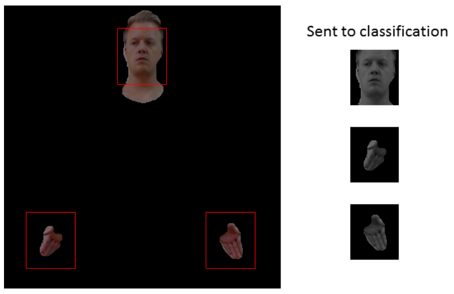


Figure 4.9: *Successful skin extraction. Equal sized boxes are sent to classification.*

4.2 Head and hand classification

In this section the results from the the head and hand classification are presented. Two types of classifiers were evaluated, SVM and CNN.

4.2.1 SVM

The best results were obtained for the SVM with a linear kernel and using HOG as feature (for more details see Section 3.3.2.1). The best validation data error rate obtained, with these parameters, was 5.0%. In Figure 4.10 a sample of images classified with the resulting SVM and their posterior probabilities are shown. As can be seen, all heads are classified correctly with rather high accuracy while one hand is misclassified. The overall accuracy of the classified hands are slightly less than that compared to the head. Moreover, the misclassification rate[3] of the samples in Figure 4.10 is 5.0%. Based on the test data, it takes on average 2.3 *ms* for the SVM to classify an image.

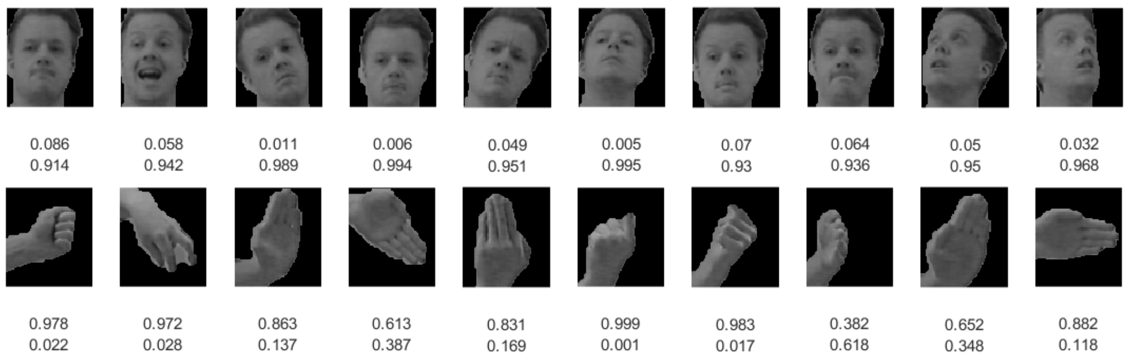


Figure 4.10: *Images classified by the produced SVM along with posterior probabilities. The upper and lower numbers represents the posterior probabilities that the image is a hand respectively a head.*

[3]Test data error rate

4.2.2 CNN

The CNN from which these results have been obtained is described in Section 3.3.2.2. With a descending learning rate of 0.0005 to 0.0003, it was possible to achieve the loss curve illustrated in Figure 4.11 after 5 epochs using a batch size of 100, for the more basic network. This CNN has the lowest validation data error rate of 1.13% and a misclassification rate of 1.645%. The resulting weights of the filters in the convolutional layers are also illustrated in Figure 4.11.

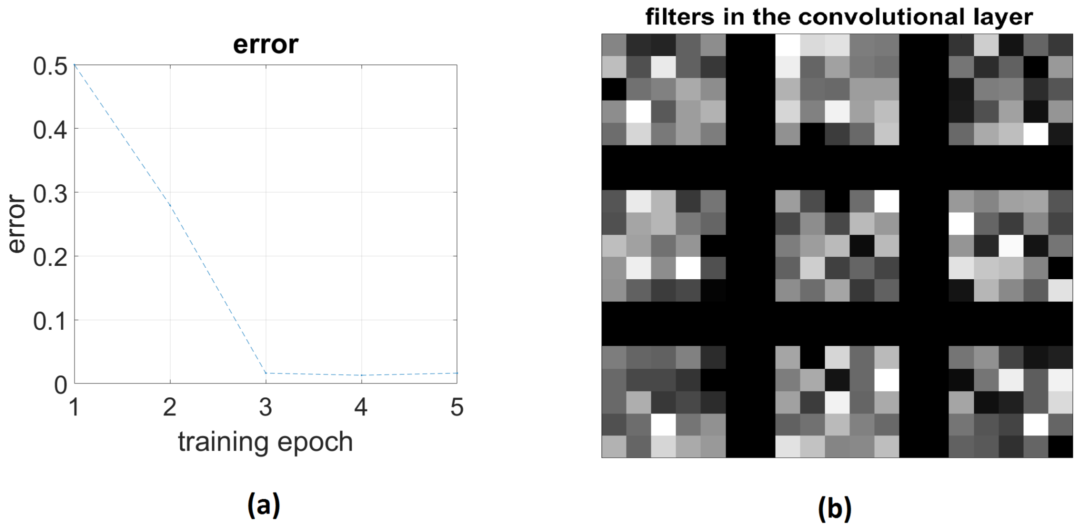


Figure 4.11: *(a) Loss curve for the basic structure of the CNN. The lowest validation data error rate is 1.13%. (b) Resulting weights of the filters in the convolutional layer for the basic structure.*

Figure 4.12 illustrates some classification examples using the basic CNN. As can be seen, three images are misclassified. However, in theory, no images should be misclassified and the misclassifications are due to the high difficulty of the images. Based on the test data, it takes on average 2.9 *ms* for the basic CNN to classify an image.

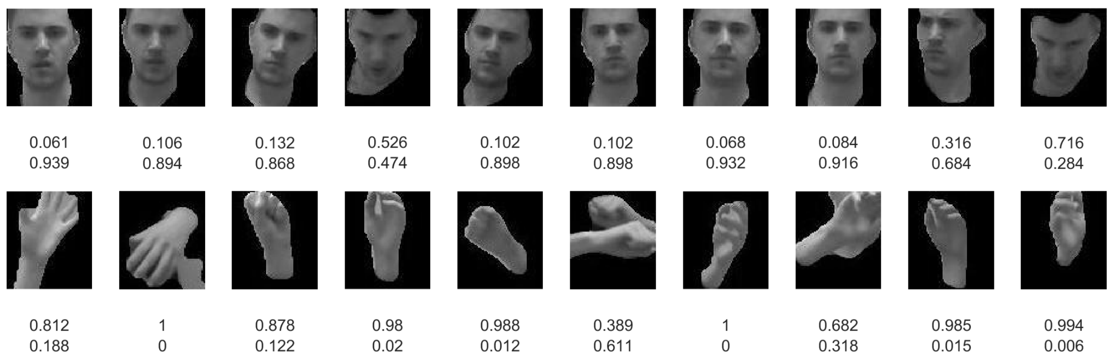


Figure 4.12: *Images classified by the produced basic CNN along with posterior probabilities. The upper and lower numbers represents the posterior probabilities that the image is a hand respectively a head.*

The more deeply structured CNN, illustrated in Figure 3.3, was trained with a descending learning rate of 0.005 to 0.0003 and produced the loss curve illustrated in Figure 4.13 after 30 epochs and a batch size of 100. The lowest validation data error rate achieved in this loss curve is 0.99% and the network has a misclassification rate of 1.645%.

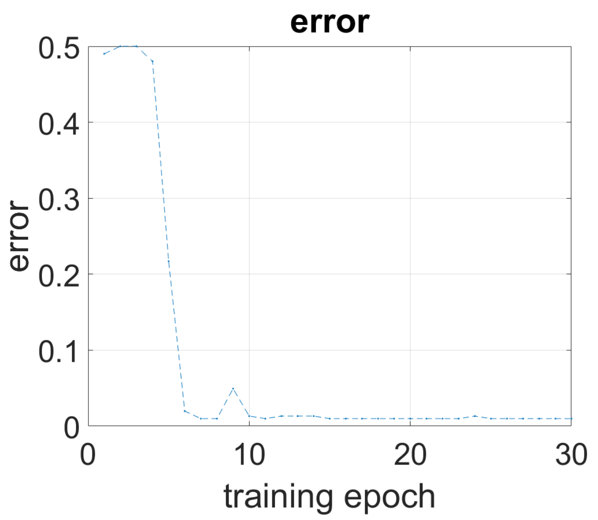


Figure 4.13: *Loss curve for the more deeply structured CNN. The lowest validation data error rate is 0.99%.*

Figure 4.14 illustrates the resulting weights of both the convolutional layers of the deeper network while Figure 4.15 illustrates some classification examples using the deeper CNN. As with the basic CNN, images are misclassified due to high difficulty. Based on the test data, it takes on average 3.9 *ms* for the deeper CNN to classify an image. Of the two CNNs, the basic is used for the classification in the final tracking system due to it being faster.

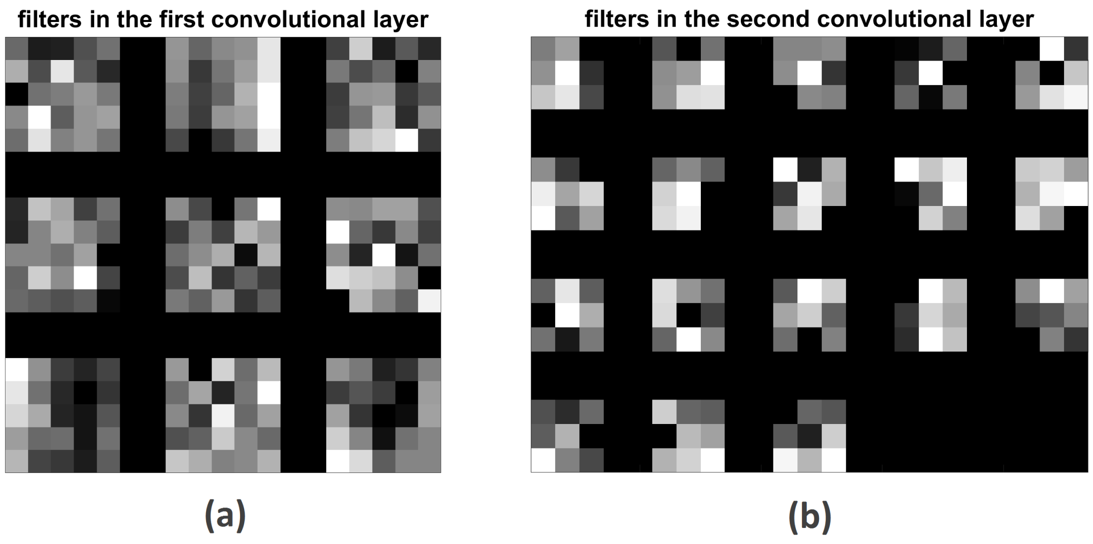

filters in the first convolutional layer
filters in the second convolutional layer
(a)
(b)

Figure 4.14: *(a) Filters in the first convolutional layer of the deeper CNN. (b) Filter in its second convolutional layer.*

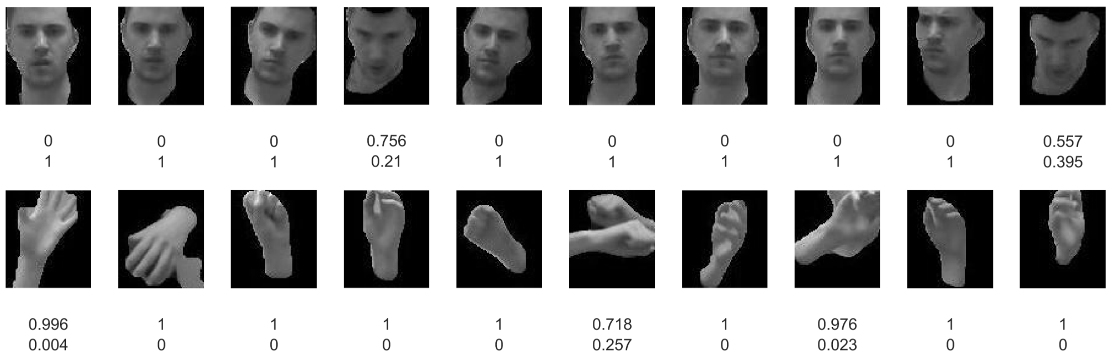


Figure 4.15: *Images classified by the deeper CNN along with posterior probabilities. The upper and lower numbers represents the posterior probabilities that the image is a hand respectively a head.*

4.3 Object tracking

The tracking used to obtain these results is described in Section 3.3.3. Figure 4.16 illustrates the sequence of how the tracking algorithm, given an image with an object to be tracked, extracts interest points and removes those that lie in the background. When the interest points have been extracted, they are tracked, as illustrated in Figure 4.17.

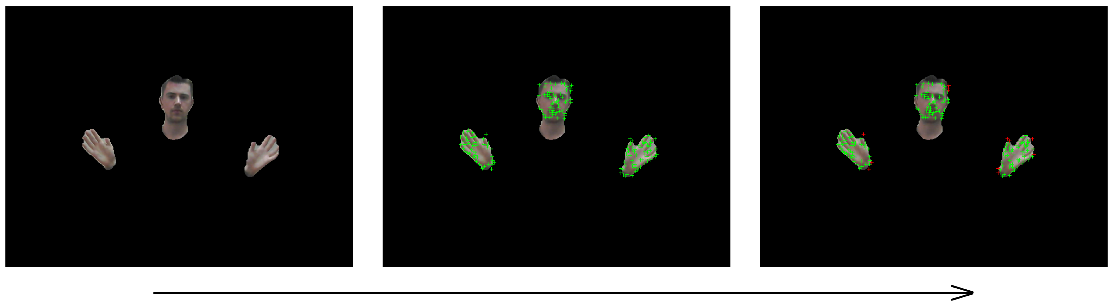


Figure 4.16: *Illustration of how interest points are extracted from the image on the left and then removed if they do not lie on the object that is being tracked. The removed interest points are marked as red in the image to the right.*

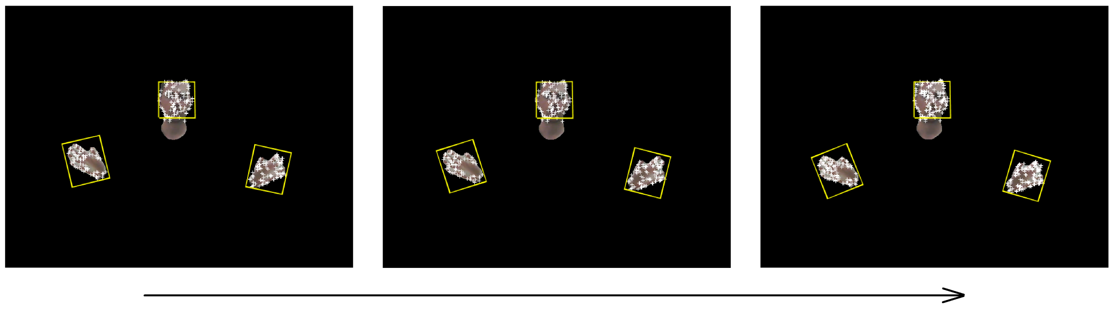


Figure 4.17: *Example of a tracking sequence over three frames. In the last frame, additional interest points are extracted to make sure that the objects are reliably tracked. The boxes surrounding each object illustrates the area where new, if any, interest points will be extracted in.*

4.4 Entire algorithm

When the entire algorithm runs it is relatively slow, as illustrated in table 4.1, in which the average execution times for each part is shown. As long as the algorithm is in the main loop in Figure 3.4 the times are stable and lower than the average. However, if it deviates from the main loop and adds more interest points or tries to detect the regions again the times are higher than the average. The average time the entire algorithm takes per frame is about 0.475 seconds.

	Background subtraction	Skin extraction	Classification	Tracking
Time (s)	0.133	0.083	0.012	0.250

Table 4.1: *Estimated execution times for the four parts of the algorithm. The time for the classification includes the time to run the CNN as well as the time to extract the correct regions. The tracking time represents the time to track all the points, remove bad points and move the boxes. The times for classification and tracking represents the time to track and classify all three objects.*

Given the execution time of the algorithm, the FPS in real time will be about two. Table 4.2 evaluates how the algorithm performs for different FPS. In order to test this, a video recorded at 30 FPS was used and to simulate a lower FPS frames were removed from the video stream accordingly. In other words, to evaluate the algorithm at 15 FPS every other frame of the 30 FPS video was removed. Clearly the algorithm is currently too slow to work properly in real time. For an FPS down to 15, the performance of the algorithm is good, but starts to get problems with certain parts of the movement below that. When the FPS is as low as 1.875 the algorithm can no longer track the body parts when they move.

FPS	Performance
30	Good
15	Good
7.5	Problems with hand twisting
3.75	Larger problems with hand twisting and objects moving fast
1.875	The tracking is completely compromised

Table 4.2: *Evaluation of how the algorithm performs on a relatively normal movement pattern. This table was created by applying the algorithm to a video recorded at 30 FPS. To simulate a lower FPS, frames were removed from the video in respective patterns.*

A big challenge to overcome was the handling of occlusions. Figure 4.18-4.20 illustrate how hand behind back, hand over head and hand over hand occlusion is handled, respectively. From the figures it can be seen that the general solution to occlusion is to keep tracking during the time of the occlusion and when the occlusion has ended (i.e. when all body parts are found again), run the classification and reinitialize trackers again.

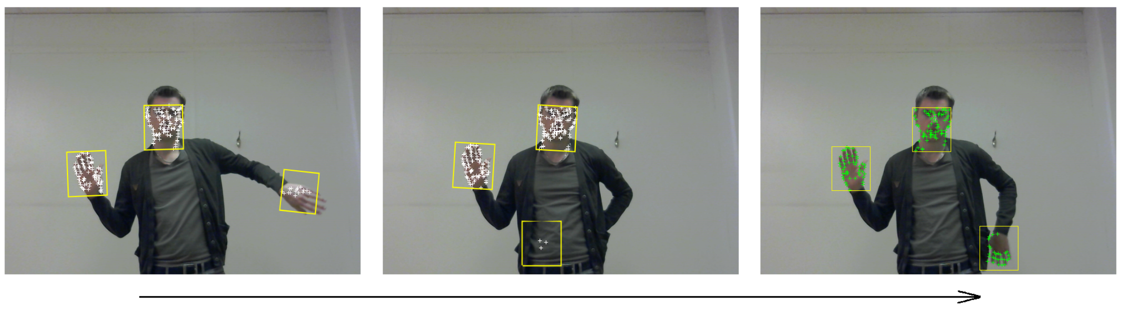


Figure 4.18: *Example showing how a hand lost behind the back and then found again. In the frame to the right, the points are green, indicating that they have all been detected in this frame.*

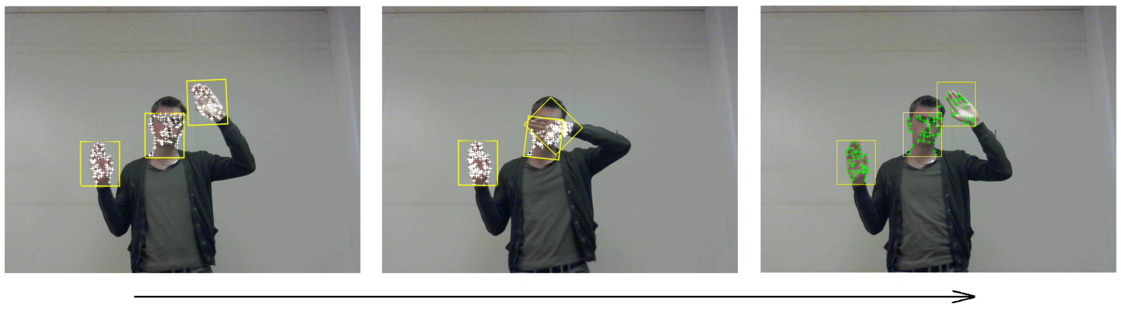


Figure 4.19: *Example showing how partial occlusion when a hand is over the head is handled. In the frame to the right, the points are green, indicating that they have all been detected in this frame.*

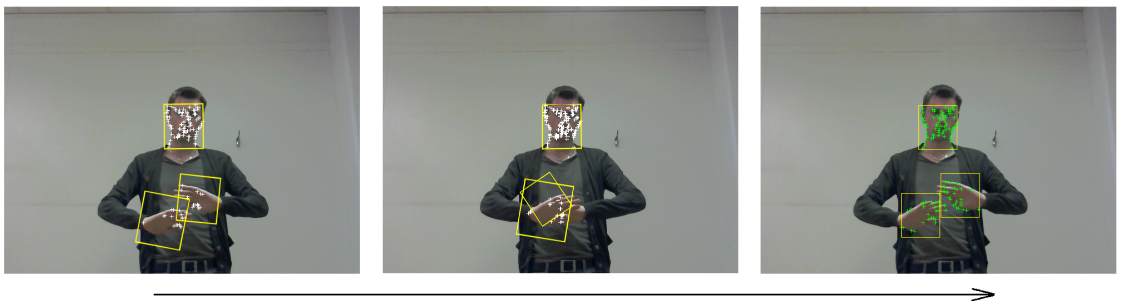


Figure 4.20: *Example showing how partial occlusion when one hand is over the other is handled. In the frame to the right, the points are green, indicating that they have all been detected in this frame.*

In appendix A longer tracking sequences on offline video recordings, meaning that a video was recorded in beforehand and then the algorithm was allowed to process each frame, can be seen. These sequences are obtained from an video recording containing 800 frames recorded at 30 FPS with a resolution of 640x480 pixels. The algorithm correctly tracks in 88.9% of these frames (meaning that in these frames the correct body parts are tracked) with a computation time of 0.475 seconds per frame. When the algorithm correctly tracks, the interest points lie exactly on the location of the head and hands, meaning that the error range is zero. This can be seen in most of the frames in A.2. However, when it incorrectly tracks as seen in frame 15 in A.2, the error range is very large. During such situations, the error range varies a lot and a value on it can not be determined.

5 Discussion

As with the results, each part of the algorithm will be discussed separately and there will be a mutual discussion for the entire algorithm. Additionally, future work and improvements on the algorithm will be discussed.

5.1 Object detection

What can be said about the detection in general is that it requires certain circumstances in order to function properly. Even if these circumstances are not present the detection is able to function but the result obtained might impede the rest of the algorithm hence lowering its performance.

5.1.1 Background subtraction

With a homogeneous background scene with good lightning conditions the performance of the background subtraction is very good. However, as background homogeneity decreases so does the performance. Basically, more objects in the background scene means a larger risk of shadows and since there is no functioning shadow removal this results in visible shadows. However, the shadow removal developed (but not implemented due to it sometimes worsening more than helping) showed promising results and it should be further developed.

Furthermore, having clothes with a color similar to that of the background will result in parts of the clothes to be considered as background. This is due to the thresholding performed. However, this is not an issue as there is no need to detect the clothes.

5.1.2 Skin extraction

Given the right preconditions the skin extraction is able to perform reasonably well. The person tracked should wear a long sleeved shirt and trousers to ensure that the only skin colored objects are the hands and the head. Also, clothes with colors similar to skin (such as yellow, pink, brown and certain shades of red) should be avoided as these colors might be extracted as well, as seen in Figure 4.8. The reason to why these colors are extracted is due to the fact that it is very difficult to isolate a single color, such as skin, while still allowing variations in it (skin color varies a lot depending on body part and lightning conditions). Hence, in order to allow variations, other colors are allowed as well. To further ease the skin extraction, skin colored objects in the background should be avoided.

By studying Figure 4.3 and 4.7 it can be seen that the skin extraction aids the background subtraction in removing background objects. Background objects not merged with the hands or the head that are left after the skin extraction should pose no threat (other than negligible time delay) in performance since these objects will, in the classification step, simply be disregarded. Examples of such objects are the objects in the lower parts of the two middle images in Figure 4.7. In both of these images four kernels will be sent for classification and the kernel containing background will be disregarded. However, in the same figure in the rightmost image, an example of an object merging with a hand can be seen. Such an event will be troublesome for the classification since the bounding box sent to classification will consist of part hand and part background resulting in a difficult classification problem.

In theory, the skin extraction should work for different lightning conditions and different skin colors due to the fact that Viola-Jones is used to, in advance, calculate the skin color. However, the algorithm has not been tested under such conditions since it has only been tested on the authors in one room.

An issue with the skin extraction step is that under some certain circumstances a hand might disappear. This is most often due to lightning. For example, if a hand is placed directly beneath a light source the color of it may become too bright hence the color is not extracted. However, as this is quite rare and since the hands are to be in almost constant movement, the hand is bound to emerge sooner or later.

5.2 Head and hand classification

The differences in classification time for the two developed CNN and the developed SVM is small. However, as expected since CNN is more complex, it has a slightly higher computation time. The simpler CNN is, on average, 0.6ms or 26% slower than the SVM. However, due to its complexity, the CNNs are able to perform slightly better with a misclassification rate of 1.645% while the SVM lies at 5% when tested on individual test data, i.e. data obtained from the author on which they were trained. Although, when tested on the same data (i.e. data from both authors), it was still found that the CNNs outperformed the SVM. For these reasons the CNN is a better choice in the classification step.

5.2.1 SVM

As mentioned above, a misclassification rate of 5.0% was obtained when using SVM. A hand can have a large number of variations in shape, in difference to a head which is quite constant, leading to data points with a wide spread in the feature space.

Therefore, if the SVM is trained with images that are not representative[1], some data points of the hand might be placed in vicinity of the data points for the head making it hard to distinguish the two sets with a hyperplane.

Even though the SVM is only trained on images from one person it should, most probably, perform equally good on other people. This since the feature chosen (HOG) is quite invariant amongst different people due to the fact that hands and heads are quite similar in their outlines.

5.2.2 CNN

Both of the CNNs created are found to have the same misclassification rate. However, the more basic one is used because it was found to be faster. Although, if it is possible to increase the speed of the system, by rewriting the code in some faster language thus making it more effective or by using a GPU, a more complex CNN could be used. This CNN could use three classes (hand, head and background) instead of just two. This would allow the system to tell if the skin and background subtraction has failed and rerun them, perhaps with different parameters, to find the hands and head.

Using the 1786 images to train, validate and test the CNNs did give satisfying results when running the entire algorithm. However, using more images could improve the misclassification rate even further. Although, since this thesis is simply a proof of concept, the 1786 images were considered to be a good representation of the different appearances of the body parts.

The loss curves for both CNNs are indeed quite strange looking, according to the theory. We believe that this is a consequence of the classification being rather simple as there are only two classes to recognize and therefore minimizing the error is a task that goes rather fast, resulting in what looks like a jump for the loss curves.

In difference to the SVM, the CNN is trained on two people which should make it even better in classifying hands and head from other people.

5.3 Object tracking

The tracking algorithm performs well given that the skin extraction only extracts the three correct objects. However, an important issue with using interest points to track objects is that the points can be blocked out of the image, such as when a hand turns and twists. Even though we try to add more interest points when some are lost, it is still difficult to track the hand when it is moving fast and twists and turns a lot. Another large issue of the KLT tracker is that it is, of course, highly dependant on the FPS of the system. We noticed that if the FPS is too low and the object moves too fast, it will loose track of it rather quickly. This is an issue

[1]Worth noting is that a large training set increase the risk of such images.

that most likely all trackers face and really the only solutions to it is to either try to make the system faster or rely more heavily upon object detection, as it is not dependant on the FPS at all (in most cases).

5.4 Entire algorithm

With an execution time of approximately 0.475 seconds per frame, the algorithm is too slow to be useful in real time. It is hard to tell if this slow execution time is due to MATLAB or due to the algorithm itself. Our theory is that is it mostly due to MATLAB since it is known to be quite a slow programming language. More specifically, it is believed that the parallel loops used in the tracking is the largest factor decreasing the speed. Therefore, to most probably increase the speed, it could be rewritten in another, faster, language such as C or C++. Without rewriting the code it is difficult to say how fast it could become and how demanding the rewriting process will be. However, we have found sources saying that the speed could be increased by a factor of 10-100 by rewriting in it C or C++. Thus, as an example, if the speed would be increased by a factor of 50, the new execution time would be around 10 milliseconds per frame resulting in that the algorithm would theoretically be able to run in 100 FPS instead of 2 FPS as it currently does. This increase in FPS would greatly improve the overall performance of the algorithm. By using MATLAB Coder, which generates C or C++ code from MATLAB code, the rewriting could be eased. Yet, by using MATLAB Coder, it is unknown how much of the rewriting would be eased since some functions and toolboxes used in the algorithm might not be supported by MATLAB Coder. Furthermore, if a real system used in rehabilitation is to be created using this algorithm, it has to be kept in mind that it should preferably work with the computing power of a tablet (since a tablet is more mobile than a laptop). However, most modern tablets have equal, or better, computing power as the laptop used in this thesis so this should not be a problem. Moreover, most tablets have cameras with a resolution better than 640x480 pixels so the resolution used in this thesis is not a problem.

The algorithm has been evaluated by using recorded videos. From this, it can be said that the algorithm, given the right circumstances, performs rather well. It is able to, in 88.9% of the frames, track the three sought parts and when it loses track of one part it is able to detect and track it again rather fast. Since we have an algorithm which detects and tracks iteratively the algorithm is able to handle such situations well. Furthermore, due to this approach, the algorithm handles occlusions such as those seen in Figure 4.18-4.20 well. This is something that is crucial for a tracking algorithm since occlusions might arise quite often. The algorithm has a hard time tracking when the hands changes appearance, if for example a hand is flipped. In such an event lots of interest points are lost due to the fact that the object now looks different from before. However, as mentioned, if a hand is lost during such an event it will eventually be found again. In recorded videos, the algorithm is able to detect both slow and fast movements of the three body parts.

In Section 1.1 it is stated that the algorithm should be able to detect the location of each body part within an error range of a few centimeters and with an accuracy of at least 90%. Furthermore, in the same section it is stated that the algorithm should have a maximum latency of 0.5 seconds. The developed algorithm has an accuracy of 88.9% and an error range of zero during these 88.9%. However, in the incorrectly tracked 11.1% the error range is large. Furthermore, the algorithm has an execution time of 0.48 seconds per frame. Thus it can be said that these goals were nearly achieved. However, an execution time of 0.5 seconds per frame and hence an FPS of 2 proved to be insufficient in order to perform a reliable tracking in real time. Regarding the accuracy and error range, the achieved results will most probably be sufficient to use in a simple rehabilitation system. This since even though the right hand might not be tracked during some frames it will eventually, rather fast if the execution time is increased, be tracked again and the user will probably not perceive this in a gaming environment.

Since the algorithm has only been tested on the authors and in one room, it is difficult to say how well it would perform at other locations and with other test subjects. In theory it should work as long as certain conditions are fulfilled. These conditions are a homogeneous background without skin colored objects, a test subject whose only visible body parts are the hands and head and who does not wear clothes with colors similar to that of skin. One disadvantage with the current Kinect-based system is that it is difficult to relocate, therefore it would be advantageous if a new system was easier to relocate. To ease relocation it would be preferred if the system could work during various circumstances, such as different backgrounds, which is something our algorithm is limited in. Hence, if this algorithm were to be used by patients it will be less user friendly than the Kinect-based system since patients would need certain clothes and a certain background. However, these conditions set aside, the algorithm can be seen as user friendly since the only thing that needs to be initialized is the capturing of images for the background model.

There are a lot of special cases that can arise during tracking. An example of such a case is the hand being split due to lightning conditions in the sequence in Figure A.2. Another example is in the same sequence where in one frame a piece of the shirt is, for some unknown reason, given a higher probability that it is a head than the actual head. Most of these events have implemented code that should deal with them. However, as the special cases differ greatly it is difficult dealing with all of them and this algorithm is not able to take care of them all. But, once again, since the algorithm detects and tracks iteratively, such situations are eventually resolved.

Since each part of the algorithm is very broad a lot of work can be put into each part to improve it. There are plenty of different approaches for each part of the algorithm and a lot of parameters to vary in each part. With this in mind, none of the parts of the algorithm has been developed to its full potential meaning that the overall performance of the algorithm is restricted. Since the parts in the algorithm depend on each other it is of importance that each part has a satisfactory performance. If this is not the case then the algorithm in whole will be impaired.

As of now, the algorithm is not optimized for use in the rehabilitation process of specifically stroke and COPD patients. Currently, the only part of the algorithm directly aimed to this rehabilitation process is the use of a non-adaptive background subtraction in order for slow or still patients to stay a part of the foreground. A limitation in this sense is the inability to differentiate between the left and right hand. This would be advantageous in a rehabilitation gaming environment since then the game could be focused on mainly performing movements with one specific hand. The reason to why this is needed is that stroke patients often have a weak side that requires more training. So to conclude, this algorithm has been developed with the purpose of tracking hands and head of a person that is not necessarily a stroke or COPD patient. However, the intended users of this algorithm are such patients. Anyhow, the developed algorithm might be of use in other areas than rehabilitation.

5.5 Comparison to Kinect

Currently, the developed algorithm is outperformed by the algorithm of the Kinect which is described in Section 2.6.3. The Kinect is able to track the entire body of the user with an error range of 1-2 cm. Our algorithm can match this error range in the 88.9% of the frames that are correctly tracked whilst in the other frames it can not. Furthermore, the Kinect is able to track at 31 FPS while our is only capable of tracking at 2 FPS. However, while our system has some limitations as to what type of background that can be used, so does the Kinect. As an example, during our master's thesis we got to try out the Kinect at a location that was somewhat crowded in the background and as a result, the Kinect had difficulties locating and tracking the user. If the FPS of our algorithm is improved it could be possible to use it as a complement to the current Kinect-based system which is used for stroke and COPD rehabilitation. For example, since the Kinect system is quite difficult to relocate and it is important that the patients perform their exercises on a regular basis it could be possible to use our algorithm to allow for patients to continue playing, somewhat simpler, rehabilitation games while they are travelling.

5.6 Contributions

Considering the first marker-less tracking solution for human-computer interaction mentioned in Section 2.7, our algorithm brings back the limitation of long sleeves and keeps the limitation of sufficient lightning condition. However, our algorithm is capable of tracking both hands and head.

The second marker-less tracking solution seems superior to our solution since it can handle cluttered environments and it does not have the limitation of long sleeves. However, it might not be well suited in the sense of a rehabilitation system.

After reviewing a lot of literature (of which some is mentioned in Section 2.7) about different approaches of creating tracking algorithms it is clear that there are extremely many ways of developing a successful tracker. Each approach has its own benefits and drawbacks. Common approaches to use are background subtraction as well as skin extraction. The use of a KLT tracker seems quite common as well. However, we found no tracking algorithm that made use of a CNN or a SVM to classify and identify the regions of interest in a real time application. Therefore, the main contribution in the field of motion tracking of this thesis is the introduction of CNN and SVM for human body tracking.

This thesis does not contribute much to the field of rehabilitation of stroke and COPD patients. This mainly due to the fact that the developed algorithm has not been tested on such patients hence no contribution can be determined. However, as mentioned in Section 1, rehabilitation based on motion tracking and playing games seems to be a successful approach. Therefore, if the developed algorithm were to be improved and implemented in rehabilitation it would most probably be able to contribute in the rehabilitation of stroke and COPD patients (and quite possibly other patients as well).

5.7 Future work

If this algorithm is to be used in a real system, future development is needed. The most important step in this future development is to rewrite the algorithm to another language in order to increase its speed. If the speed is satisfactory after such a rewriting, the algorithm could be further developed and improved. Except from rewriting the algorithm, one way that might improve the speed (which has not been examined) is to lower the resolution and hence the amount of pixels that are to be processed. However, lowering the resolution also lowers the amount of detail in the image leading to less accuracy. If the speed can not be increased, this algorithm will not be of any use in a real system.

The speed set aside, more work can be put into each part of the algorithm. Of most importance is to improve the background subtraction and skin extraction since if these fail, the following steps will have a hard time performing well. The main thing to focus on in these steps is to implement a functioning shadow removal as well as, if possible, improve the skin extraction so that less colors are extracted. To prevent situations when the hand is split into two, as seen in Figure A.2, the skin extraction could make it easier for pixels adjacent to extracted regions to be classified as skin. For example, if half a hand has been extracted the other half could be found by examining the neighborhood (in the original image) with a lower requirement on the color. Furthermore, the algorithm should be tested on persons with a different skin color than the authors in order to see if the skin extraction works as it theoretically should. If the background subtraction and skin extraction are improved and the speed is increased, a usable algorithm might very well be achieved.

There is also more work needed on the object tracking and classification. As mentioned in the discussion, the biggest problem that the tracking faces is when the hands change appearance, hence more work is needed to try to improve the tracking during such events. One solution to this could be to extract more interest points more often. Currently, in the developing world of CNNs, a new, extremely fast, network type has been created [47]. The type is called fully convolutional network and is specialized at semantic segmentation, which means segmenting the image into different regions and classifying each of these regions. It is capable of doing this for entire images 114 times faster than the state of the art while keeping a comparable accuracy. Using such a network, hands and head could possibly be classified faster, without the need for background subtraction and skin extraction. An alternative to a fully convolutional network is to add a new class to the current CNN. This class should be background and hence the CNN will be able to differentiate between hand, head and background and not just hand and head. In order to do this there is a need to train the CNN on a large amount of different backgrounds. If successful, such an approach will be able to work as a safety net if the skin extraction has failed and bounding boxes containing background or a combination of background and hand have been sent to the classification step.

If the algorithm is to be implemented in a real system used in rehabilitation there is also a need for the algorithm to be able to differentiate between the left and right hand as previously discussed. This could be achieved if we have a CNN that is trained to do so. However, it is possible that such a network would need to be deeper than the one we currently use to achieve a satisfying misclassification rate.

6
Conclusion

This thesis has developed a tracking algorithm able to track the hands and head of a person. The algorithm is divided into three major parts, detection, classification and tracking. Detection is done with background subtraction and skin extraction. The result from this step is, ideally, an image containing solely two hands and one head. A CNN is used for classifying objects in the image obtained in the detection step and when classification has been completed a KLT tracker is used to track the three body parts.

The developed algorithm is too slow to work in real time having an execution time of around 0.48 seconds per frame. Moreover, to fully be able to perform the algorithm needs some preconditions on the environment as well as on the person. Since the algorithm is slow it is troublesome to evaluate the performance (speed set aside) of it. However, by allowing the algorithm to run on recorded videos it can be said that it performs rather well with an accuracy of 88.9%. There is still a lot of future work to be done on all the parts of the algorithm in order for it to be implemented in a system used for rehabilitation. Particularly, there is a need to make the algorithm faster. This can most probably be done by rewriting the algorithm in a faster language.

To conclude this report, consider the key questions in Section 1.1. The approach used makes the algorithm able to, with the right preconditions, quite accurately track the movement of the hands and head. However, work is needed to improve accuracy. Regarding efficiency, since the algorithm is unable to track in real time, it fails. As of now, the algorithm is not able to achieve similar results as the algorithm used for tracking in the Kinect. This is mainly due to its slow speed but also due to its restricted performance. It will probably be possible to increase the speed and hence the performance of the algorithm, as previously discussed, but the algorithm will probably still be inferior to the Kinect tracking algorithm. This does not come as a surprise since the Kinect tracking algorithm has been developed by a huge company during a large time span and with the use of a depth camera while this work has been a simple proof of concept during a few months. However, as discussed in Section 5.5, if the speed of the developed algorithm is increased it might be possible to use it as a complement to the Kinect-based system.

Bibliography

[1] Socialstyrelsen. (n. d.). Statistikdatabas för stroke, [Online]. Available: http://www.socialstyrelsen.se/statistik/statistikdatabas/stroke. [Collected: 10 feb. 2016].

[2] Hjärt-Lungfonden. (2010). Stroke, [Online]. Available: https://www.hjart-lungfonden.se/Documents/Skrifter/Skrift_stroke_2012.pdf. [Collected: 10 feb. 2016].

[3] A. Stuart, "Arm and hand exercises for stroke rehab", *WebMD*, 2010. [Online]. Available: http://www.webmd.com/stroke/features/arm-and-hand-exercises-for-stroke-rehab, [Collected: 21 dec. 2015].

[4] Hjärt-Lungfonden. (Aug. 2010). Hjärt-Lungfondens KOL-rapport, [Online]. Available: https://www.hjart-lungfonden.se/Documents/Rapporter/KOL-rapporten%202012.pdf. [Collected: 12 feb. 2016].

[5] STROKE - Riksförbundet. (n. d.). Rehabilitering, [Online]. Available: http://www.strokeforbundet.se/show.asp?si=460&sp=442&go=Rehabilitering. [Collected: 12 feb. 2016].

[6] KOL. (n. d.). Träning vid kol, [Online]. Available: http://www.kol.se/diagnos-behandling/traening-vid-kol/. [Collected: 21 dec. 2015].

[7] NeuroOptima. (2016). Vår rehabiliteringsmetod, [Online]. Available: http://www.neurooptima.com/se/metod-och-forskning/neurorehabilitering-med-spraktraning/. [Collected: 19 aug. 2016].

[8] H. Sveistrup, "Motor rehabilitation using virtual reality", *Journal of NeuroEngineering and Rehabilitation*, vol. 1, no. 1, pp. 10–17, Dec. 2004. DOI: 10.1186/1743-0003-1-10.

[9] R. Kizony, N. Katz, P. L. Weiss, "Adapting an immersive virtual reality system for rehabilitation", *The Journal of Visualization and Computer Animation*, vol. 14, no. 5, pp. 261–268, Dec. 2003. DOI: 10.1002/vis.323.

[10] K. Bower *et al.*, "Clinical feasibility of interactive motion controlled games for stroke rehabilitation", *Journal of NeuroEngineering and Rehabilitation*, vol. 12, no. 1, pp. 63–77, 2015. DOI: 10.1186/s12984-015-0057-x.

[11] R. Wardini *et al.*, "Using a virtual game system to innovate pulmonary rehabilitation: Safety, adherence and enjoyment in severe chronic obstructive pulmonary disease", *Canadian Respiratory Journal*, vol. 20, no. 5, pp. 357–361, Sep. 2013.

[12] A. Yilmaz, O. Javed, M. Shah, "Object tracking: A survey", *ACM Computing Surveys (CSUR)*, vol. 38, no. 4, 13–es, Dec. 2006. DOI: 10.1145/1177352.1177355.

[13] A. Smeulders, R. Cucchiara, A. Dehghan, “Visual tracking: An experimental survey”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 7, pp. 1442–1468, Jul. 2014. DOI: `10.1109/TPAMI.2013.230`.

[14] H. Bay *et al.*, “Speeded-Up Robust Features (SURF)”, *Computer Vision and Image Understanding*, vol. 33, no. 16, pp. 2094–2101, Dec. 2012. DOI: `10.1016/j.cviu.2007.09.014`.

[15] S. S. Cheung, C. Kamath, “Robust techniques for background subtraction in urban traffic video”, *Visual Communications and Image Processing*, vol. 5308, no. 1, pp. 881–892, Jan. 2004. DOI: `10.1117/12.526886`.

[16] P. Soille, *Morphological Image Analysis: Principles and Applications*. Springer Berlin Heidelberg, 2004. DOI: `10.1007/978-3-662-05088-0`.

[17] SDSU Library. (n. d.). Understanding dilation and erosion, [Online]. Available: `http://www-rohan.sdsu.edu/doc/matlab/toolbox/images/morph4.html`. [Collected: 1 July 2016].

[18] Kungliga Tekniska Högskolan. (n. d.). Median filtering, mode filtering, and rank leveling, [Online]. Available: `http://medim.sth.kth.se/6l2872/F/F7-1.pdf`. [Collected: 22 june 2016].

[19] C. Jung, “Efficient background subtraction and shadow removal for monochromatic video sequences”, *IEEE Transactions on Multimedia*, vol. 11, no. 3, pp. 571–577, 2009. DOI: `10.1109/TMM.2009.2012924`.

[20] J. M. Chaves-González *et al.*, “Detecting skin in face recognition systems: A colour spaces study”, *Digital Signal Processing*, vol. 20, no. 3, pp. 806–823, May 2010. DOI: `10.1016/j.dsp.2009.10.008`.

[21] MathWorks. (n. d.). Supervised learning, [Online]. Available: `http://se.mathworks.com/discovery/supervised-learning.html`. [Collected: 4 april 2016].

[22] I. Steinwart, A. Christmann, *Support vector machines*, 1:st; ser. Information science and statistics. New York: Springer, 2008.

[23] beaglef tk, *Support Vector Machines Kernels I*, YouTube, 2014. [Online]. Available: `https://www.youtube.com/watch?v=HwQQXs4Nyjo`, [Collected: 26 may 2016].

[24] R. Nogueira, R. Alencar Lotufo, R. Campos Machado, “Fingerprint liveness detection using convolutional neural networks”, *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 6, pp. 1206–1213, 2016. DOI: `10.1109/TIFS.2016.2520880`.

[25] K. O’Shea, R. Nash, “An introduction to convolutional neural networks”, *CoRR*, vol. abs/1511.08458, 2015. DOI: `arXiv:1511.08458`.

[26] A. Karpathy. (n. d.). Convolutional neural networks for visual recognition (CNNs / ConvNets), [Online]. Available: `http://cs231n.github.io/`. [Collected: 18 May 2016].

[27] S. Bell. (n. d.). Convolutional neural networks, [Online]. Available: `http://www.cs.cornell.edu/courses/cs4670/2015sp/lectures/lec32_cnns_web.pdf`. [Collected: 19 may 2016].

[28] S. Warren, C. Sarle. (Mar. 2014). What is a softmax activation function?, [Online]. Available: `http://www.faqs.org/faqs/ai-faq/neural-nets/part2/section-12.html`. [Collected: 19 may 2016].

[29] N. Srivastava *et al.*, “Dropout: A simple way to prevent neural networks from overfitting”, *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[30] D. Wilson, T. Martinez, “The general inefficiency of batch training for gradient descent learning”, *Neural Networks*, vol. 16, no. 10, pp. 1429–1451, 2003. DOI: `10.1016/S0893-6080(03)00138-2`.

[31] Welch Labs, *Neural Networks Demystified [Part 2: Forward Propagation]*, YouTube, 2014. [Online]. Available: `https://www.youtube.com/watch?v=UJwK6jAStmg`, [Collected: 01 July 2016].

[32] ——, *Neural Networks Demystified [Part 3: Gradient Descent]*, YouTube, 2014. [Online]. Available: `https://www.youtube.com/watch?v=5u0jaA3qAGk`, [Collected: 01 July 2016].

[33] ——, *Neural Networks Demystified [Part 4: Backpropagation]*, YouTube, 2014. [Online]. Available: `https://www.youtube.com/watch?v=GlcnxUlrtek`, [Collected: 01 July 2016].

[34] A. Y. Ng *et al.* (n. d.). Ufldl tutorial, [Online]. Available: `http://ufldl.stanford.edu/tutorial/`. [Collected: 30 june 2016].

[35] C. Tomasi, T. Kanade, “Shape and motion from image streams: A factorization method - Part 3: Detection and tracking of point features”, Computer Science Department, Pittsburgh, PA, Tech. Rep. CMU-CS-91-132, Apr. 1991.

[36] S. U. E. Hildreth, “The measurement of visual motion”, *Trends in Neurosciences (Regular ed.)*, vol. 6, pp. 177–179, Jan. 1983. DOI: `10.1016/0166-2236(83)90081-4`.

[37] D. Navneet, B. Triggs, “Histograms of oriented gradients for human detection”, *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 886–893, 2005. DOI: `10.1109/CVPR.2005.177`.

[38] National Instruments. (Jun. 2015). Feature extraction, [Online]. Available: `http://zone.ni.com/reference/en-XX/help/372916T-01/nivisionconcepts/feature_extraction/`. [Collected: 8 June 2016].

[39] P. Viola, M. Jones, “Rapid object detection using a boosted cascade of simple features”, *Computer Vision and Pattern Recognition*, vol. 1, pp. 511–518, 2001. DOI: `10.1109/CVPR.2001.990517`.

[40] J. MacCormick. (2011). How does the Kinect work?, [Online]. Available: `http://users.dickinson.edu/~jmac/selected-talks/kinect.pdf`. [Collected: 10 june 2016].

[41] A. Mobini, S. Behzadipour, M. S. Foumani, “Accuracy of Kinect’s skeleton tracking for upper body rehabilitation applications”, *Canadian Respiratory Journal*, vol. 9, no. 4, pp. 344–352, 2014. DOI: `10.3109/17483107.2013.805825`.

[42] H.-S. Yeo, B-G. Lee, H. Lim, “Hand tracking and gesture recognition system for human-computer interaction using low-cost hardware”, *Multimedia Tools and Applications*, vol. 74, no. 8, pp. 2687–2715, 2015. DOI: `10.1007/s11042-013-1501-1`.

[43] CM. Huang, YR. Chen, LC. Fu, “Visual tracking of human head and arms using adaptive multiple importance sampling on a single camera in cluttered

environments", *IEEE Sensors Journal*, vol. 14, no. 7, pp. 2267–2275, Jul. 2014. DOI: 10.1109/JSEN.2014.2309256.

[44] Mathworks. (n. d.). Image processing toolbox, [Online]. Available: http://se.mathworks.com/products/image/. [Collected: 22 june 2016].

[45] Lenovo. (n. d.). Lenovo g50-80, [Online]. Available: http://shop.lenovo.com/us/en/laptops/lenovo/g-series/g50-80/#tab-tech_specs. [Collected: 20 jan. 2016].

[46] Logitech. (n. d.). Hd webcam c270, [Online]. Available: http://www.logitech.com/sv-se/product/hd-webcam-c270. [Collected: 3 mars 2016].

[47] J. Long, E. Shelhamer, T. Darrell, "Fully convolutional networks for semantic segmentation", *Computer Vision and Pattern Recognition*, pp. 3431–3440, Jun. 2015. DOI: 10.1109/CVPR.2015.7298965.

A

Tracking sequences

Figure A.1 shows nine frames from a video sequence. The sequence is recorded in 30 FPS and in the figure each tenth frame is shown meaning that the time difference between each frame is 1/3 of a second. The sequence shows an event with rather fast movement and also an event in which the hand drastically changes appearance. As can be seen in the fourth frame there are very few interest points left, however, in the fifth frame more have been added and therefore the algorithm is able to handle the situation.

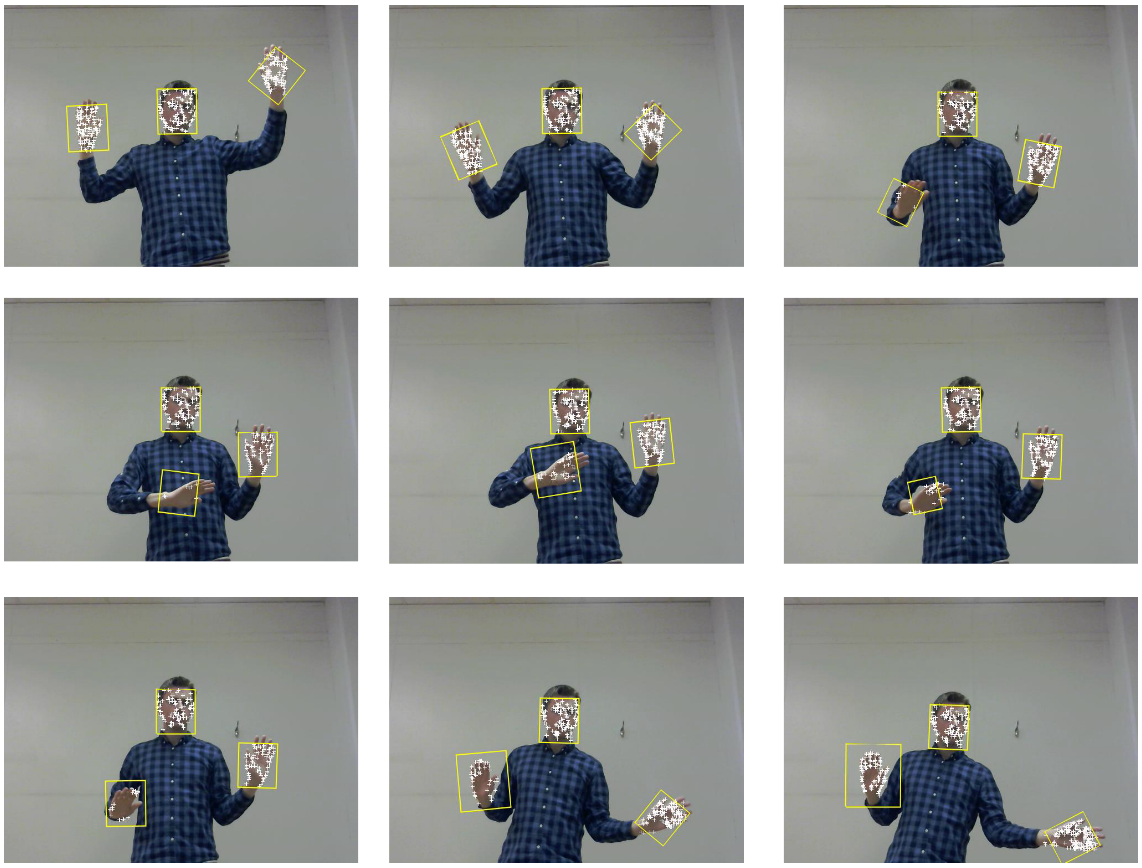
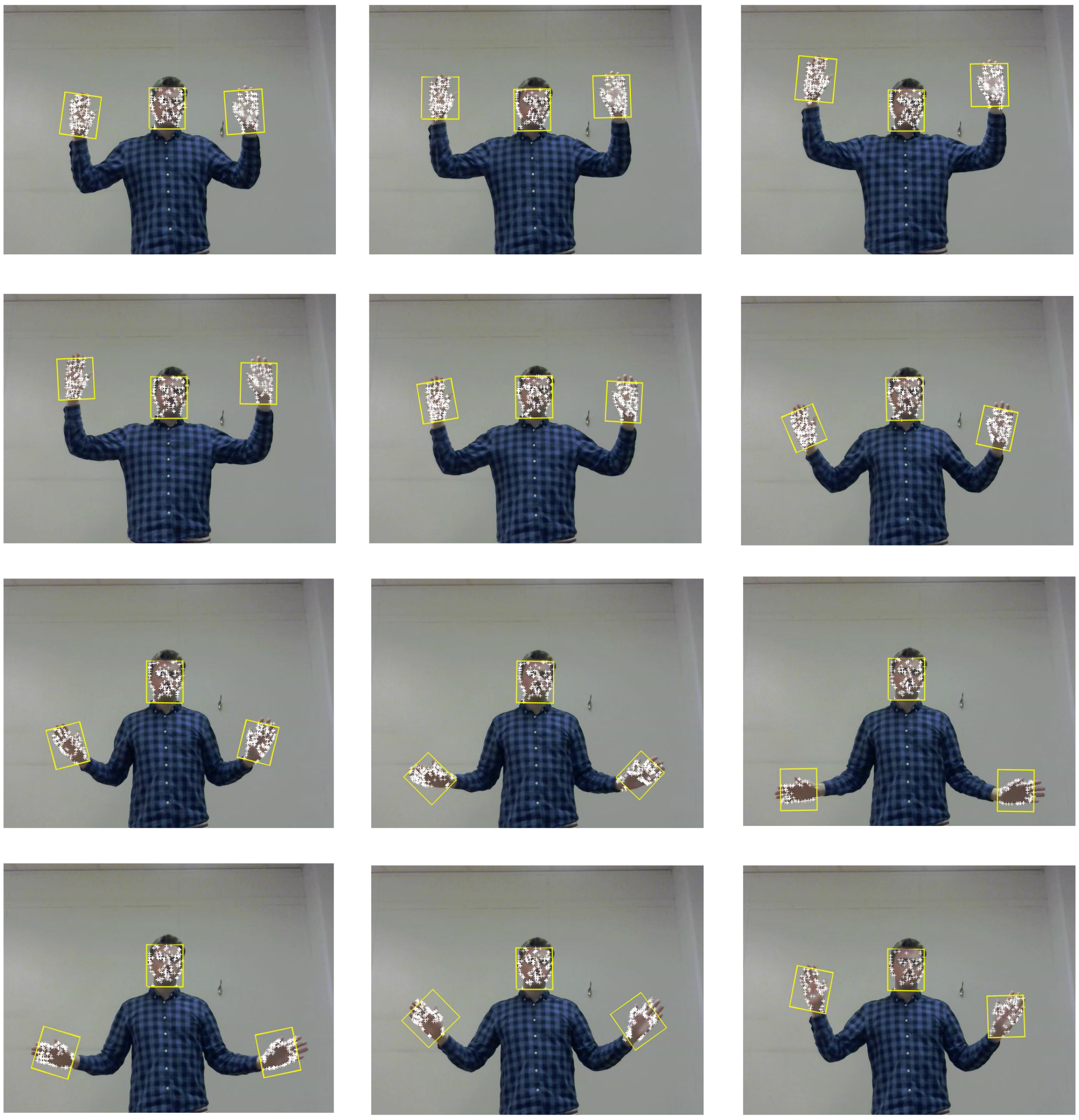


Figure A.1: *Video sequence recorded with 30 FPS. White crosses represent interest points.*

Figure A.2 shows 24 frames from a video sequence. As with the first sequence it is recorded in 30 FPS with each tenth image shown. In this sequence the movement is slower than in the previous sequence. In the fifteenth and sixteenth image of the sequence, the left hand is detected as two separate objects resulting in that two parts of that hand is tracked and no part of the right hand is tracked. This is due to a faulty skin extraction and such events should be dealt with as described in Section 3.3.1.2. However, as seen, it is not working in this sequence. In the seventeenth frame, a piece of the shirt has passed through the skin extraction (probably due to shadows) and is hence a candidate for classification. It has in this frame been faulty classified with a higher score than the real head thus it is considered to be a head and is therefore tracked. As can be seen, eventually this faulty tracking is resolved and the correct objects are tracked in the eighteenth frame.



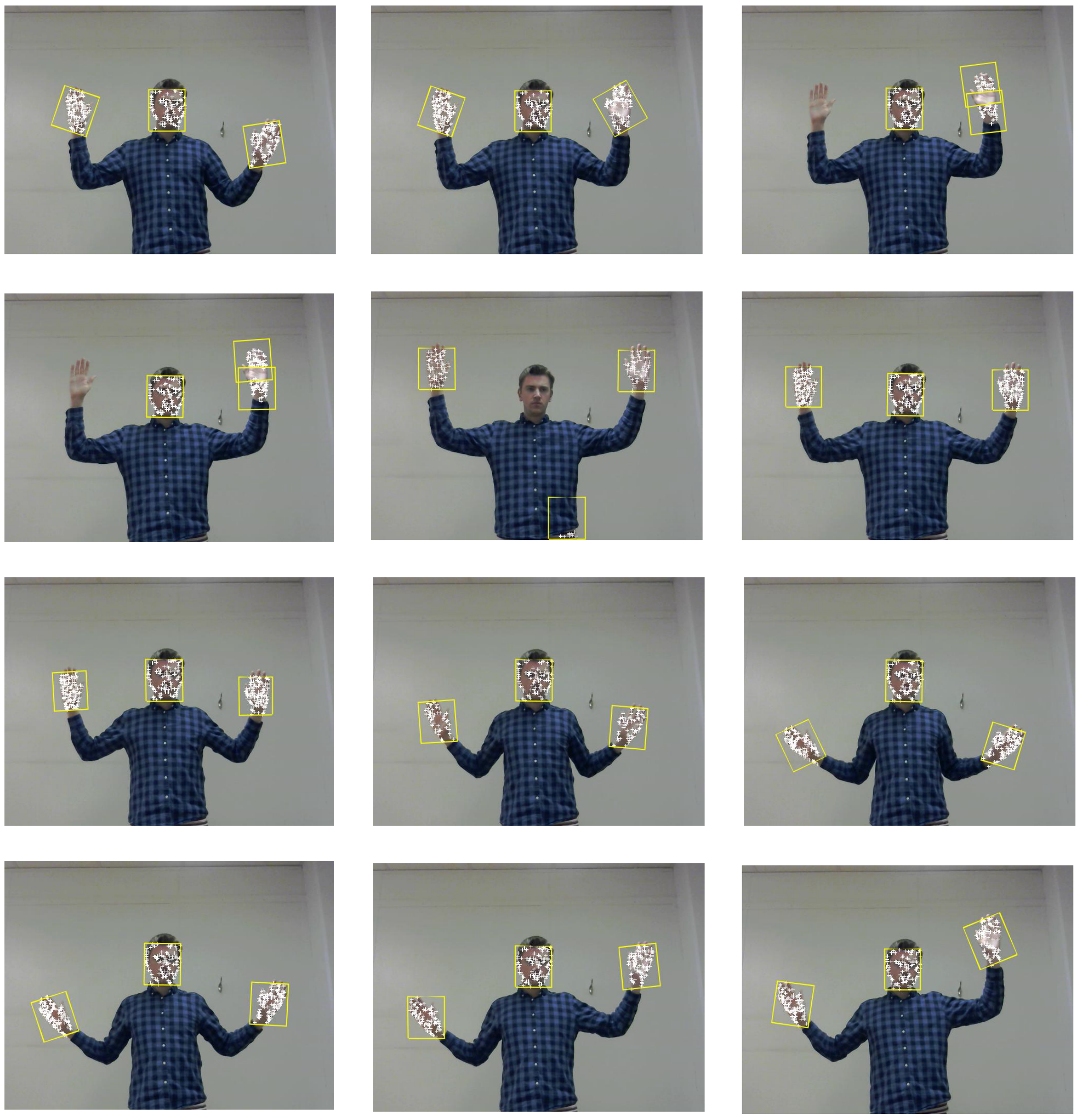


Figure A.2: *Video sequence recorded with 30 FPS. White crosses represent interest points.*

B

Threshold levels

B.1 Background subtraction

In table B.1 the threshold levels used in the background subtraction can be seen. Y, Cb and Cr ranges from 0-255. The thresholds work so that if the difference in Y-, Cb- and Cr-value between the background model and current scene at a certain pixel is less than 16, 6 and 4, respectively, then the pixel is determined to belong to the background. If not, the pixel is determined to be in the foreground. The values were experimentally obtained using independent data set and a single environment. The criterion was that all potential foreground pixels should be detected, i.e. a high sensitivity to the user.

Color	Value
Y	16
Cb	6
Cr	4

Table B.1: *Threshold levels used in the background subtraction.*

B.2 Skin extraction

In table B.2 the threshold levels used in the skin extraction can be seen. R, G and B ranges from 0-255 and H, S and V ranges from 0-1. A pixel is determined to be of skin color if all of its color components (R, G, B, H, S and V) are within the low and high values given in table B.2. The values were experimentally obtained using independent data set and a single environment. The criterion was that all potential skin pixels should be detected, i.e. a high sensitivity to skin.

Color	Low	High
R	0	-
G	30	-
B	10	-
H	$\bar{H}$ - 0.2	$\bar{H}$ + 0.2
S	$\bar{S}$ - 0.2	$\bar{S}$ + 0.3
V	$\bar{V}$ - 0.3	$\bar{V}$ + 0.3

Table B.2: *Threshold levels used in the skin extraction. $\bar{H}, \bar{S}$ and $\bar{V}$ are the mean values on the skin color obtained from the Viola-Jones algorithm.*

Acronyms

CNN - Convolutional Neural Network
COPD - Chronic Obstructive Pulmonary Disease
GLCM - Gray-Level Co-occurrence Matrix
HOG - Histogram of Oriented Gradients
KLT - Kanade-Lucas-Tomasi
LBP - Local Binary Pattern
SVM - Support Vector Machine