Friedrich-Alexander Universität Erlangen-Nürnberg

Chair of Multimedia Communications and Signal Processing

Prof. Dr.-Ing. André Kaup

Bachelor Thesis

Facial Landmark Detection in Teleconference Environments

Mohamed Krissaane

March 2019

Supervisor: Kristian Fischer M.Sc

Acknowledgements

I confirm that I have written this thesis unaided and without using sources other than those listed and that this thesis has never been submitted to another examination authority and accepted as part of an examination achievement, neither in this form nor in a similar form. All content that was taken from a third party either verbatim or in substance has been acknowledged as such.

_____ _____

Ort, Datum Unterschrift

Lehrstuhl für Multimediakommunikation
und Signalverarbeitung
Prof. Dr.-Ing. A. Kaup

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Bachelor's Thesis

for

Mr. Mohamed Krissaane

Facial Landmark Detection in Teleconference Environments

Erkennung von Gesichtsmerkmalen in Telekonferenzszenarien

Nowadays, more and more meetings are hold via the internet. Accordingly, there is a strong need for applications that allow to transmit the participant's current webcam picture to the receiver's side. In order to save bandwidth, special video codecs are required that are capable of coding the information in the most effective way and under real-time constraints. Clearly, the most important area of a webcam image is the face of the participant. Therefore, one can extract the facial landmarks from the input frame and use them for efficient coding.

The task of this research internship is to find and evaluate algorithms for facial landmark detection already known in the literature. Some of the most significant and promising algorithms are supposed to be embedded into a testing and evaluation framework. Finally, the performance of the selected facial landmark detection algorithms shall be evaluated in terms of reliability and computation time. Besides, the algorithms should be optimized for teleconference applications, preferably with 4K data.

The preferred programming language of the algorithms is C++. A well-structured implementation and a detailed documentation of the algorithms and experiments is part of the thesis.

Start Date: 05.11.2018

Finish Date: 05.04.2019

A. Kaup

(Prof. Dr.-Ing. A. Kaup)

Contents

Kurzfassung

Heutzutage werden immer mehr Konferenzen und Meetings über das Internet durchgeführt. Die Übertragung des Videostroms vom Sender auf die Empfängerseite, insbesondere bei Verwendung hochauflösender Kameras, erfordert eine große Bandbreite und ist zeit- und energieaufwendig. Daher sind geeignete Anwendungen erforderlich, um den wesentlichen Bereich eines Video-frames, nämlich das Teilnehmersgesicht, effizient und schnell auf die andere Seite zu übertragen. Diese Arbeit präsentiert vier verschiedene Gesichtserkennungsfunktionen, Haar-like features (HAAR), Local binary pattern (LBP), Deep Neutral Network (DNN) und Histogram of oriented gradient (HOG) aus den Bibliotheken Open Source Computer Vision Library (OpenCV) und DLIB. Die Funktionalität dieser Funktionen und die Schritte, die sie zur Erkennung der Gesichter durchführen, werden im ersten Abschnitt vorgestellt. Anschließend wird ein allgemeiner und detaillierter Einblick in die Implementierung der verwendeten Algorithmen gegeben. Die in dieser Arbeit behandelten Merkmale haben ihre enthaltenen Parameter; die Rolle der einzelnen Parameter wird im zweiten Abschnitt beschrieben. Zusätzlich und für weitere Tests und Verbesserungen werden HAAR, LBP, DNN ,und HOG durch verschiedene Testsätze geführt, nämlich Parameter-Tweaking, bei dem die Parameter der On-Features geändert und modifiziert werden, Bildvorverarbeitung, bei der die Eingangsdaten modifiziert werden, bevor sie der Erkennungsfunktion zugeführt werden, und Algorithmus-Kombination, bei der ein paar Algorithmen kombiniert werden, um die Erkennungsleistung zu verbessern. Nach der Prüfung werden die Ergebnisse für die vier Modelle verglichen und im vierten Teil der Arbeit bewertet. Die Experimente zeigten, dass die Vorverarbeitungstechnik verschiedene Auswirkungen

auf die Leistung der Merkmale hat. DNN und HOG lieferten die besten Ergebnisse unter Berücksichtigung der Genauigkeit und der Schnelligkeit der Erkennung, insbesondere nach der Größenänderung der Vorverarbeitungstechnik für DNN und der Entrauschungstechnik für HOG.

Abstract

Nowadays more and more conferences and meetings are being held over the internet. The transmission of the video stream from the sender to the receiver side, and especially when using high-resolution cameras, needs a wide bandwidth and is time and energy consuming. Therefore, appropriate applications are necessary to transmit the essential region of a frame, which is the participant's face, to the other end efficiently and rapidly. This Thesis presents four different face detection features, HAAR, LBP, DNN, and HOG from OpenCV and DLIB libraries. The functionality of these features and the steps they deploy to detect the faces will be presented in the first section. Then a general and detailed look at the implementation of the used algorithms will be shown. The features handled in this thesis do have their included parameters; the role of each parameter is described in the second section. Additionally and for further testing and improving, HAAR, LBP, DNN, and HOG are passed through different sets of test namely parameter tweaking, where the feature parameters are changed and modified, image preprocessing, where the input data is modified before feeding it to the detection function, and algorithm combination, where a couple of algorithms are combined together to enhance the detection performance. After testing, the results for the four models are compared and evaluated in the fourth part of the thesis. The experiments showed that the preprocessing technics has various impacts on the performance of the features. DNN and HOG delivered the best results considering the accuracy and the rapidity of detection especially after the resizing preprocessing technic for DNN and the denoising technic for HOG.

Acronyms

OpenCV Open Source Computer Vision Library

LBP Local binary pattern

Adaboost Adaptive Boosting

DNN Deep Neutral Network

HAAR Haar-like features

HOG Histogram of oriented gradient

SVM support vector machine

ADT Average detection time

BLAS Basic Linear Algebra Subprograms

LPF Low Pass Filter

FPS Frame rate per second

Operators and Symbols

$+$	Addition
$-$	Substarction
$*$	Multiplication
$I(x)$	Pixel intensity of x
$\sqrt{}(x)$	Square root of x
$\|x\|$	Absolute value of x

Chapter 1

Introduction

During the last decade, the researches in the face detection area became more and more active, and it is today one of the relevant research topics in the field of biometrics. The reason is that there are many applications of this task such as face recognition, video conferences, content-based image retrieval, video surveillance, human-computer interface, face expressions analysis, visitors counting, access control where the detection is the first stage of any face processing. The face detection task is quite easy for humans but becomes a serious problem while developing an automatic detection system. In this case, it is necessary to deal with many factors which affect the facial appearance such as viewpoint, slope, face expression, occlusions, light conditions, etc. but with machine learning, pattern recognition, artificial intelligence, and computer vision technology continue to develop, face recognition technology has made significant progress.
This thesis presents four well-known face detection methods: HAAR, HOG, LBP, and DNN which introduced into preprocessing and combination process in the purpose of improving the features potential and to determine the best detection technic.

Chapter 2

Algorithm introduction

2.1 OpenCV

OpenCV is an open source computer vision and machine learning software library. OpenCV was developed by Intel and built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products. Being a BSD-licensed product, OpenCV makes it easy to utilize and modify the code [Ope19].

OpenCV is written in C++ and its primary interface is in C++, but it still retains a less comprehensive though extensive older C interface. There are bindings in Python, Java and MATLAB/OCTAVE.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to provide a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality.

2.1.1 HAAR

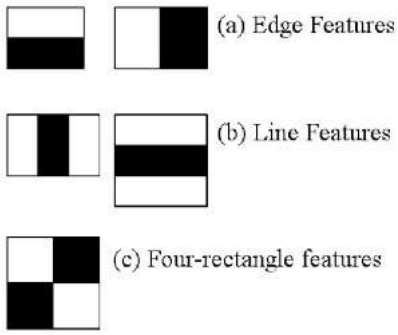


Figure 2.1: Some HAAR-like features [VJ11]

Object Detection using HAAR feature-based cascade classifiers is an object detection method proposed by Paul Viola and Michael Jones in their paper [VJ11]. It is a machine learning based approach where a cascade function is trained from many positive and negative images. Positive images are the ones containing the object that is needed to be detected, which is the face, and the negative ones are the images that do not contain faces in them. After the training, it is then used to detect faces in other pictures.
The feature-based method locates faces by extracting structural features of the face as the features shown in Figure 2.2. These features consist of black and white pixels, and they are divided into subcategories depending on the positioning of these pixels such as edge features, line features, etc.. (Figure 2.1).
Figure 2.2 shows a person's face. It is easy to perceive that the nose region has three different subregions; brighter pixels in the middle and darker ones on the left and the right, and this is, for example, where the vertical line feature can be applied.
The example shown in the Figure 2.2 is rather an ideal HAAR feature extraction considering that in a real image the intensity of the pixels is between 0 (white) and 1 (black) (Figure 2.3) and can in very rarely be 0 and 1.

In this case, a Δ value is calculated to determine if the group of pixels represents a HAAR feature. In 2.1, the average intensity under the white rectangles is subtracted

(a) Input image (b) Detected line feature

Figure 2.2: HAAR feature extraction

0	0	1	1
0	0	1	1
0	0	1	1
0	0	1	1

(a) Ideal HAAR-feature pixel intensities

0.2	0.1	0.6	0.8
0.2	0.1	0.8	0.6
0.3	0.3	0.6	0.9
0.1	0.2	0.6	0.8

(b) Real detected intesity values

Figure 2.3: Difference between an ideal and a real HAAR feature

from the black rectangle, where n is the number of pixels on the black side, m is the number of pixels on the white side, and $I(i)$ is the intensity of the corresponding pixel.

$$\Delta = 1/n \sum_{i=1}^{n} I(i) - 1/m \sum_{i=1}^{m} I(i) \tag{2.1}$$

The Δ from Figure 2.3b is equal to :

$$\Delta = 0.7125 - 0.1875 = 0.525 \tag{2.2}$$

(a) Different kernel positions (b) Different kernel sizes

Figure 2.4: Different kernel's position and sizes

and the closer Δ to 1, which is the Δ value of ideal HAAR-feature pixel intensities, the more likely that the pixels represent a HAAR-feature.

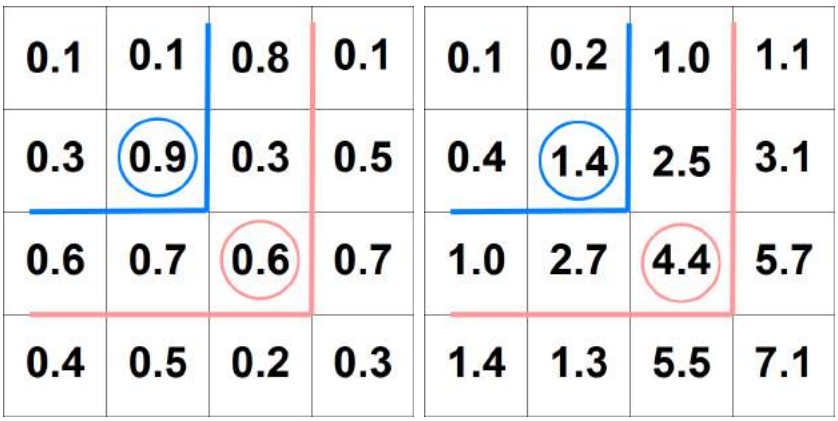
The Viola Jones calculation utilizes a 24×24 window as the base window size to begin evaluating these features in any given image, and considering all possible parameters of the HAAR features like position, scale and type as shown in Figure 2.4 , it is expected to calculate Δ for more than 160,000 features in this window, accordingly the calculation of Δ can be time consuming.

A solution to this problem was proposed by Viola and Jones [VJ11] by transforming the original image to an integral image. This transformation is done by calculating a new value for each pixel which is the sum of pixels above and to the left of a given pixel (Figure 2.5).

The advantage of this transformation is that the Integral image allows for the calculation of the sum of all pixels inside any given rectangle using only four values at the corners of the rectangle (The circled values in Figure 2.6).

The sum value in the given box is then calculated like follow:

$$Sum = 5.5 - 1 + 0.1 - 1.4 = 3.2 \tag{2.3}$$



(a) Original image (b) Integral image

Figure 2.5: Integral image transformation

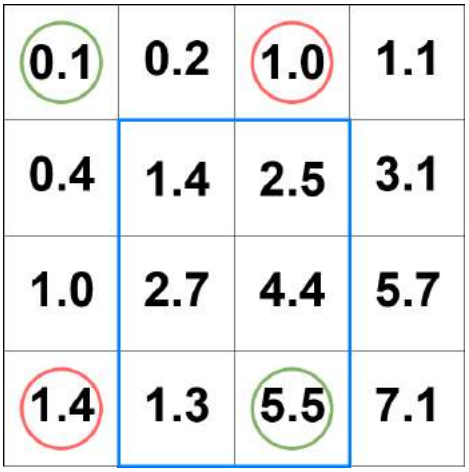
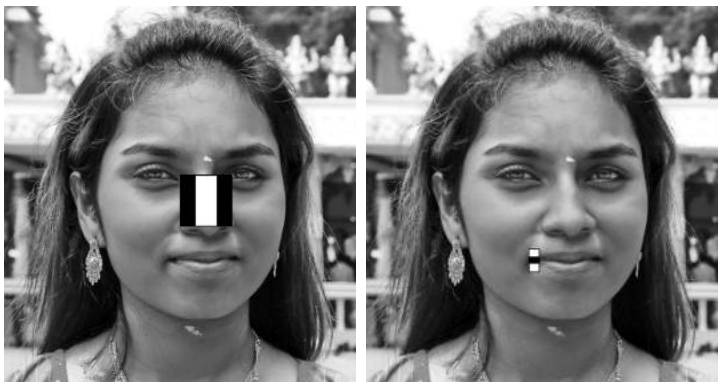


Figure 2.6: Original image

As stated previously approximately 160,000 feature values can be within a detector at 24×24 base resolution, but only a few sets of features will be useful among all these features to identify a face. Figure 2.7 shows the difference between a useful and an unuseful feature.

To reduce the number of features and keep just the important ones, Viola and Jones used Adaptive Boosting (Adaboost). AdaBoost is a machine learning algorithm formulated by Yoav Freund and Robert Schapire [Ada19]. AdaBoost is adaptive in the sense that subsequent weak learners (Features) are tweaked in favor of those instances misclassified by previous classifiers.



(a) Useful feature (b) Unuseful feature

Figure 2.7: Difference between useful and unuseful features

With the help of Adaboost, the features or weak classifiers $f_{\mathrm{i}}(x)$ are then linearly combined to construct a strong classifier $F(x)$ (2.4).

$$F(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + ... \tag{2.4}$$

Despite using AdaBoost and decreasing the number of features to approximately 6,000, the number of features is still too hight to perform rapid face detection. Concentrating on the fact that some of these features are more informative than others, Viola and Jones [VJ11] created a new concept called a cascade of classifiers. This process is composed of multiple stages, and all the features are grouped in several stages starting by the most important ones. Using this method will detect the nonfacial regions and discard them without consuming more time by passing them to the next steps 2.8.

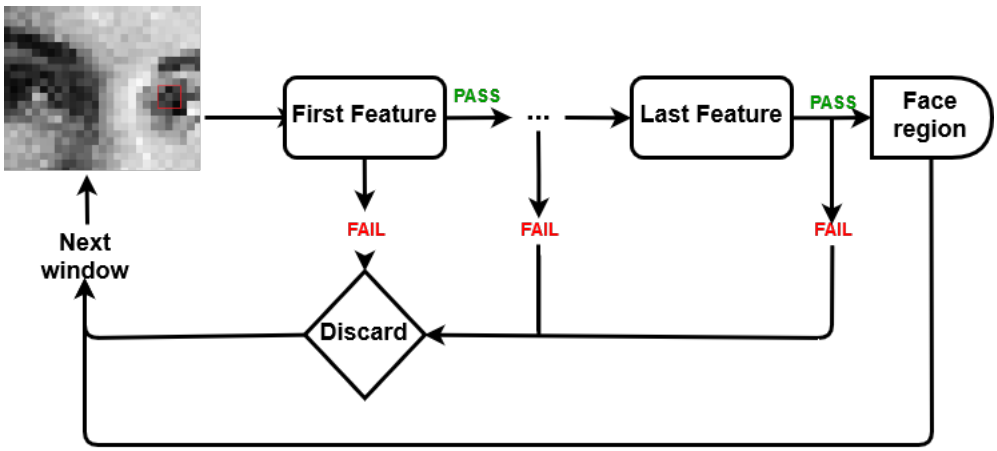


Figure 2.8: Cascade classifier

2.1.2 LBP

LBP is a texture descriptor made famous by the work of Ojala et al. in their 2002 paper, [Pra17] (although the concept of LBPs was introduced as early as 1993).
The original LBP operator, introduced by Ojala [TOH94] characterizes the neighborhood of a point of an image in a 3×3 window by calculating the difference of the gray level of the pixel considered which is the middle pixel with gray levels of pixels in the neighborhood. For each neighbor pixel that is greater than or equal to the center pixel, the value will be set to 1, otherwise 0 (2.5). The new values will be read in a clockwise order starting with the top left corner, and the result will be considered as a binary number.

$$s(x) = \begin{cases} 1, & \text{if } x \geq 0. \\ 0, & otherwise. \end{cases} \tag{2.5}$$

An illustration of the basic LBP operator is shown in Figure 2.9.

The most prominent limitation of the original LBP operator is its small spatial support area. Features calculated in a local 3×3 neighborhood cannot capture large scale

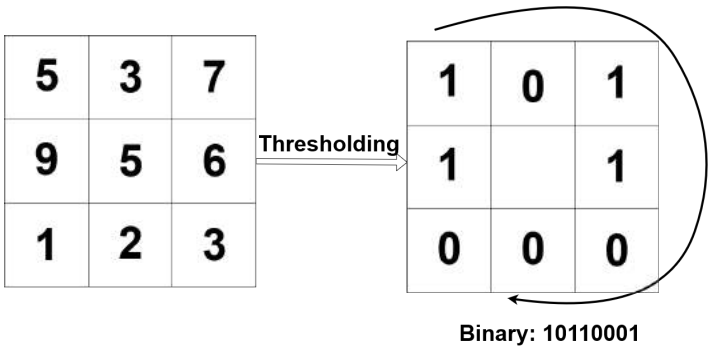
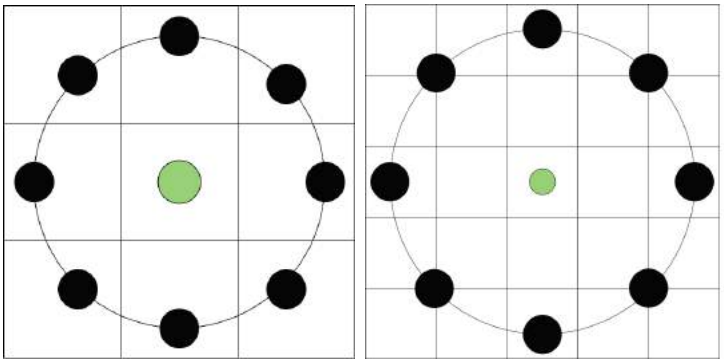


Figure 2.9: Labeling the pixels of an original image



(a) The circular (8,1) neighborhood

(b) The circular (8,2) neighborhood

Figure 2.10: Two examples of extended LBP

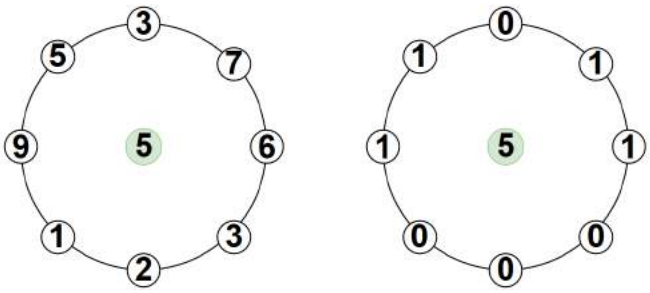
structure that may be the dominant features of some textures. For this reason, the LBP operator was extended to use neighborhoods of different sizes (Ojala et al. 2002 [Pra17]). Using circular neighborhoods and bilinearly interpolating the pixel values allow any radius and number of pixels in the neighborhood [Pra17] (Figure 2.10).

In the following, the notation (N,R) will be used for pixel neighborhoods which means N sampling points on a circle of radius of R. The value of the LBP code of pixel (x_c, y_c) is given by:

$$LBP_{\mathrm{N,R}}(x,y)\sum_{i=0}^{N-1} = s(n_c - n_i)2^i \ with \ s(x) = \begin{cases} 1, & \text{if } x \geq 0. \\ 0, & otherwise. \end{cases} \tag{2.6}$$

Figure 2.11 shows an example of LBP computation.

The value of the middle point will then be replaced by the new decimal number from



(a) Sample (b) Threshold

Figure 2.11: An example of LBP computation

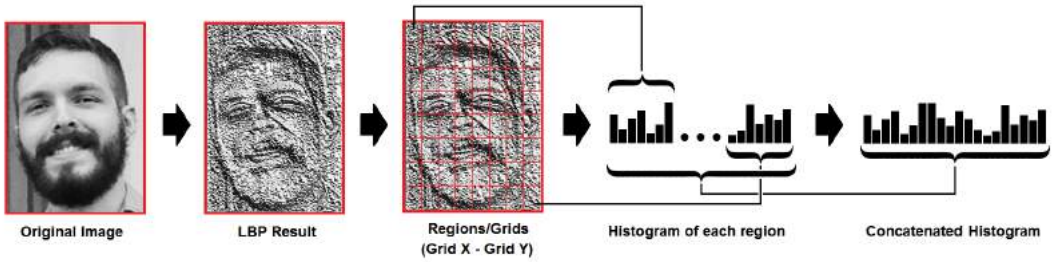


Figure 2.12: Extracting a histogram from the original image [TOM02]

the binary number (Figure 2.9). The LBP image will be divided into blocks; each block will be then converted to a histogram. The histograms will be joined together and create a histogram for the whole image (Figure 2.12).

The histogram of the input image is then compared with the histograms of the training images using the Euclidean distance.

$$D = \sum_{i=0}^{n} = \sqrt{(hist1_{\mathrm{i}} - hist2_{\mathrm{i}})^2} \tag{2.7}$$

The calculated distance is used then as a confidence threshold that is used to estimate if the algorithm has successfully detected a face.

2.1.3 DNN

Deep Learning is the most popular and the fastest growing area in Computer Vision nowadays. DNN was included in OpenCV version 3.3. In contrast with image processing using HAAR, LBP or HOG, the feature extraction and classification with DNN is done by the neural net.

A neural net consists of different layers; the input layer, the hidden layers, and the output layer.

The input layer is formed by input neurons which corresponds to the pixels of an input image. To calculate the value of the neurons in the second layer R_i (first hidden layer), weights w_i are assigned to the connection between the current neuron and the previous neurons (input neuron). The intensity value of the input neurons is multiplied with their associated weights to build a weighted value. A bias value, which can be described as an offset value, is added to the weighted value and considering that the values of the new neurons of the next layers cannot exceed -1 or 1, a sigmoid function is used.

$$R_\mathrm{i} = S((\sum_{i=1}^{n} I_\mathrm{i} * w_\mathrm{i}) + b) \tag{2.8}$$

OpenCV provides multiple models for this face detector such as Caffe, Tensorflow, etc.. where the weights of the actual layers are stored.

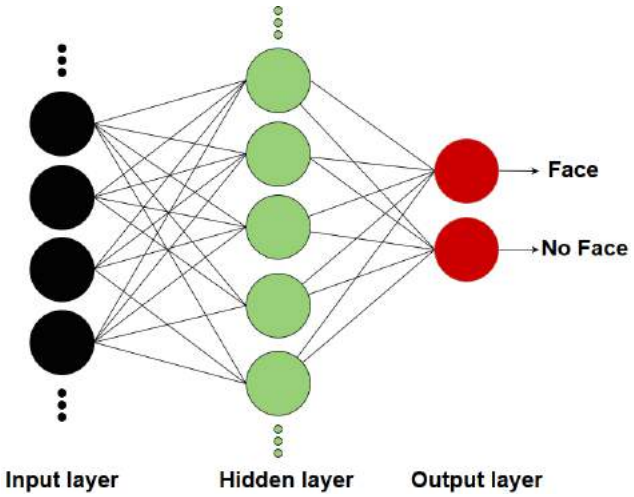


Figure 2.13: Neural network layers

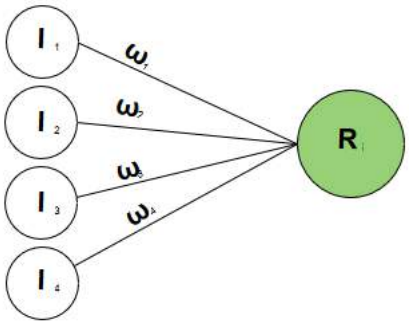


Figure 2.14: Assign weights

2.2 DLIB

Dlib is a general purpose cross-platform software library written in the programming language C++. It is used in both industry and academia in a wide range of domains including robotics, embedded devices, mobile phones, and large high-performance computing environments. Thus it is, first and foremost, a set of independent software components. It is open-source software released under a Boost Software License.
The core of the Dlib is linear algebra with Basic Linear Algebra Subprograms (BLAS) and the library has two main components, linear algebra, and machine learning tools [Kin09b].
Since development began in 2002, Dlib has grown to include a wide variety of tools.
As of 2016, it contains software components for dealing with networking, threads, graphical user interfaces, data structures, linear algebra, machine learning, image processing, data mining, XML and text parsing, numerical optimization, Bayesian networks, and many other tasks [DLI19].
In recent years, much of the development has been focused on creating a broad set of statistical machine learning tools and in 2009 Dlib was published in the Journal of Machine Learning Research [Kin09a].

2.2.1 HOG

Dalal et al. proposed HOG in 2005 [DT05] to detect pedestrians and it is a feature descriptor used in computer vision and image processing for object detection.
The technique counts occurrences of gradient orientation in localized portions of an image. This method is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy, and it often used with support vector machine (SVM) classifiers.
Figure 2.15 shows the main steps for face detection using the HOG feature.

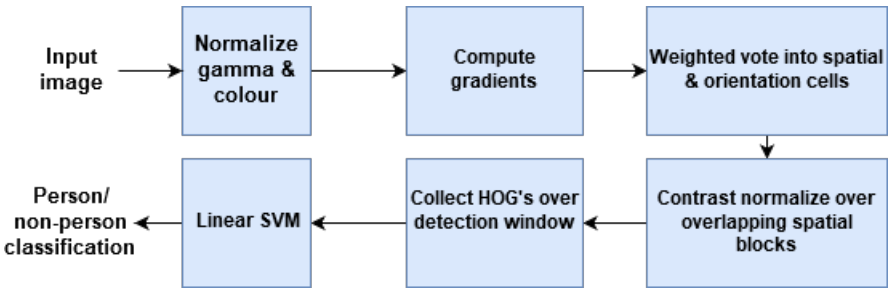


Figure 2.15: HOG feature detection steps [DT05]

The Normalization step is optional, and it has three main methods:

Gamma/power law normalization: In this case, the $\log(p)$ of each pixel p in the input image is taken. However, as Dalal and Triggs demonstrated [DT05], this approach is perhaps an over-correction and hurts performance.

Square-root normalization: Here, the $\sqrt{p}$ of each pixel p in the input image is taken. By definition, square-root normalization compresses the input pixel intensities far less than gamma normalization. And again, as Dalal and Triggs showed in their paper [DT05], square-root normalization increases accuracy rather than hurts it.

Variance normalization: A slightly less used form of normalization is variance normalization. Here, both the mean μ and standard deviation σ of the input image are computed. All pixels are mean centered by subtracting the mean from the pixel intensity, and then normalized through dividing by the standard deviation: $p' = (p - \mu)/\sigma$. Dalal and Triggs do not report accuracy on variance normalization.

Then from each pixel, the gradient components (x,y) in both horizontal and vertical directions are extracted using 2.9 and 2.10.

In addition to the vertical and the horizontal directions, HOG also needs the magnitude and the direction of the gradient of each pixel point, that can be calculated using 2.11 and 2.12.

$$Horizontaldirection : G_{\mathrm{x}}(x, y) = I(x + 1, y) - I(x - 1, y) \tag{2.9}$$

$$Vertical\ direction : G_{\mathrm{y}}(x, y) = I(x, y + 1) - I(x, y - 1) \tag{2.10}$$

$$Gradient\ magnitude : m(x, y) = \sqrt{(G_{\mathrm{x}}(x, y)^2) + (G_{\mathrm{y}}(x, y)^2)} \tag{2.11}$$

$$Gratient\ direction : \theta(x, y) = \frac{G_{\mathrm{x}}(x, y)}{G_{\mathrm{y}}(x, y)} \tag{2.12}$$

Given both $|G|$ and θ, the histogram of oriented gradients can be now computed, where the bin of the histogram is based on θ and the contribution or weight added to a given bin of the histogram is based on $|G|$.

HOG feature will divide the input image into small windows; each window consists of an 8×8 pixel cell. Taking an example of a 64 window, this window will be divided into an eighty 8 pixel cells. In each cell, the magnitude values of these 64 cells are binned (Figure 2.16) and cumulatively added into 9 buckets of unsigned direction corresponding to angles $0, 20, 40..160$ (Figure 2.17).

Focusing on the pixel encircled in blue, it has an angle (direction) of 80 degrees and a magnitude of 2. So it adds 2 to the 5th bin. The gradient at the pixel encircled using red has an angle of 10 degrees and a magnitude of 4. Since 10 degrees is halfway between 0 and 20, the vote by the pixel splits evenly into the two bins.

The contributions of all the pixels in the 8×8 cells are added up to create the 9-bin histogram. The final histogram is shown in Figure 2.18.

Then in the normalization step, a 2×2 cells (thus 16x16 pixels) block is slid across the image by 8 pixels each time as shown in Figure 2.19. In each block region, four histograms of 4 cells are concatenated into a one-dimensional vector of 36 values and then normalized to have a unit weight so that the histogram will be immune to the lighting variations.

The final HOG feature vector is the concatenation of all the block vectors, and it can

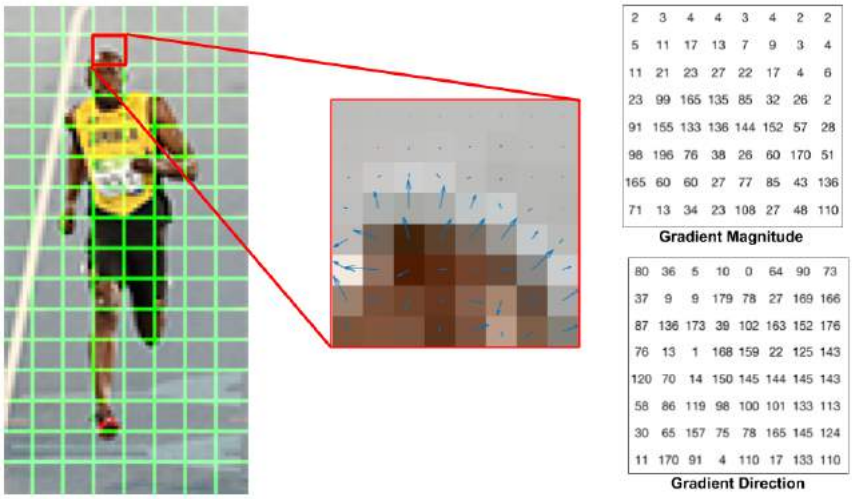


Figure 2.16: Gradient magnitude and direction

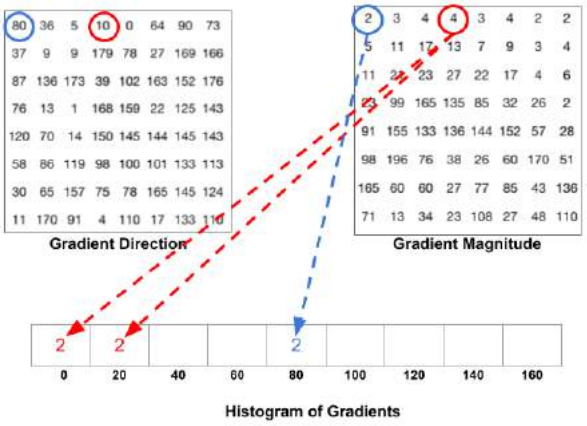


Figure 2.17: HOG histogram creation

be fed into a classifier like SVM for learning object recognition tasks. The Figure 2.20 shows the HOG output.

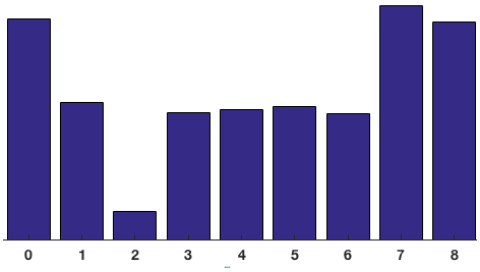


Figure 2.18: HOG histogram

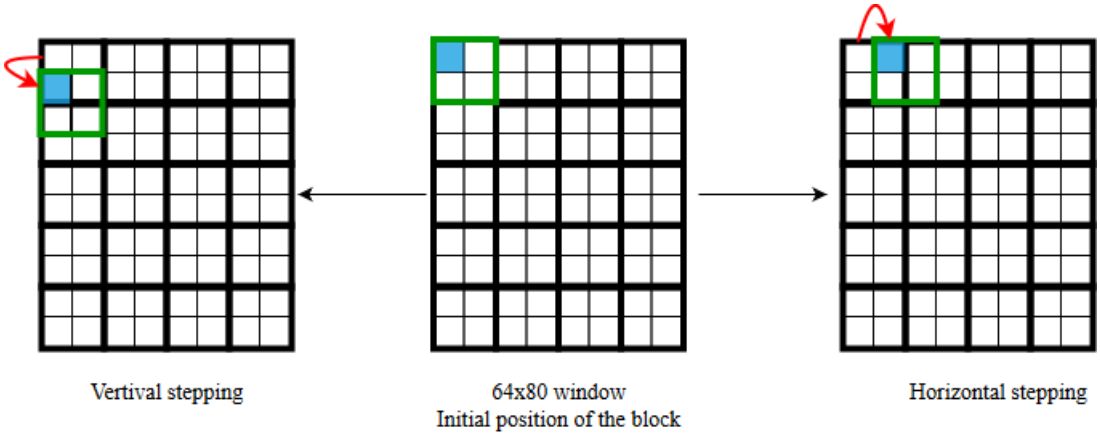


Figure 2.19: HOG: Stepping process for the block

Figure 2.20: HOG image [HOG16]

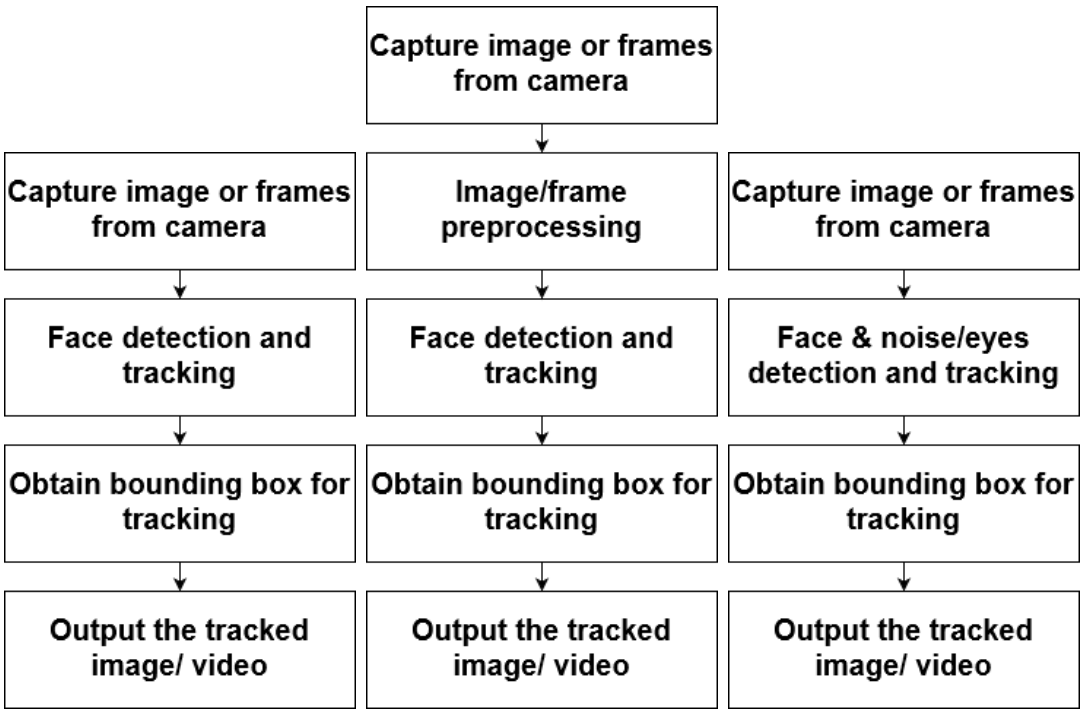
Chapter 3

Implementation and Tracking Algorithm

3.1 Algorithm structure

The algorithms proposed in this thesis have the same structural steps. Figure 3.1a shows a basic algorithm detection structure, which composes of four main phases; Image or video frames reading, feed the images to the detection and tracking function and if a face was detected a bounding box will be drawn around the face and the resulted image will be then given as an output.
In section 4.1, the parameters of the primary detection function in step two will be varied and tested. HAAR and LBP have the same settings listed below:

ScaleFactor : Parameter determining how much the image size is reduced at each image scale. Mainly, the scale factor is utilized to make a scale pyramid. In short, the model has a fixed size defined during training, which is visible in the .xml file. This means that this size of the face is detected in the image if present.
However, by rescaling the input image, a larger face can be resized to a smaller one, making it detectable by the algorithm.



(a) Basic face detection steps (b) Image preprocessing before detection (c) Several facial landmarks detection

Figure 3.1: Different implementations steps

Taking 1.05 as an example, which means using a small step for resizing, i.e., reduce the size by 5%, increases the chance of finding more faces. This also means that the algorithm works slower since it seeds to go through more layers. **minNeighbors :** Parameter specifying how many neighbors each candidate rectangle should have to retain it. This parameter will affect the quality of the detected faces.

minSize : Minimum possible object size. Objects smaller than that are ignored. This parameter determines how small size we want to detect. Usually, [30, 30] is a good start for face detection.

maxSize : Maximum possible object size. Objects bigger than this are ignored. This parameter determines how big size we want to detect.

Usually, it isn't needed to set it manually, the default value assumes it is wanted to detect without an upper limit on the size of the face. In my case, I didn't define it either.

DNN also has these parameters:

ScaleFactor : After performing mean subtraction, the images can be optionally scaled by some factors. This value defaults to 1.0 (i.e., no scaling) .
size : Here the spatial size that the Convolutional Neural Network expects is supplied. In other words, the size of the input image or the window, where the faces are being detected. This parameter will be tested in the resolution test section 4.2.2.
mean : These are the mean subtraction values. They can be a 3-tuple of the RGB means, or they can be a single value in which case the supplied value is subtracted from every channel of the image.
In short, mean subtraction means helping to fight with illumination changes in images and thus helping the Convolutional Neural Networks. The mean values are used from ImageNet training set, and values are for RED=123.68, GREEN=116.77, and BLUE=103.93 channels.
swapRB : OpenCV assumes images are in BGR channel order; however, the mean value assumes the channel is in RGB order. To resolve this discrepancy, swap the R and B channels in an image by setting this value to True is needed. By default, OpenCV performs this channel swapping automatically.
In addition to these parameters, DNN has a Confidence threshold. If the system is confident, that the object detected is a face, is higher than the confidence threshold, then it will be detected as a human face. If not, it will be neglected.

$$confidence > confidenceThreshold \tag{3.1}$$

And the last technic is HOG, which has only a scale factor parameter.
For further testing and to improve the algorithm output results, an two extended version of the primary detection steps is implemented. In Figure 3.1b the input data is first preprocessed. Image processing is a method to perform some operations on the images, to get an enhanced image or to extract some useful information from it and in

Figure 3.2: Image after grayscaling

this case the face features.

This Thesis presents multiple image preprocessing technics :

Grayscale : The role of grayscale is to transform the input picture or video frames to a black and white frame 3.2. it is a transformation within RGB space like removing the alpha channel, reversing the channel order, conversion from 16-bit RGB color (R5:G6:B5 or R5:G5:B5), as well as conversion to grayscale, using:

$$RGB\ to\ Gray: \quad Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{3.2}$$

Resizing : The images used in the analysis are resized in different scales to determine how various sizes affect the recognition process (Figure 3.3). Different image sizes carry different information that is why the best image size needs to be examined in details. The purpose of image resizing is to produce a lower data size, which hastens the processing time.

Filter and denoising : Images are often by default have Gaussian noise due to illumination variations. In the experiment section, a Gaussian blur filter which is a Low Pass Filter (LPF) is used to eliminate high-frequency information and retain only with low-frequency information. The Gaussian blur is a type of image-blurring filter that uses a Gaussian function (which also expresses the normal distribution in statistics)

Figure 3.3: Image after Resizing

Figure 3.4: Image after Gaussian Blur

for calculating the transformation to apply to each pixel in the image [Gau18]. The formula of a Gaussian function in one dimension is

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \tag{3.3}$$

In two dimensions, it is the product of two such Gaussian functions, one in each dimension:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.4}$$

Figure3.4 shows the difference between noisy image and filtered image.

Brightness : Two commonly used point processes are multiplication and addition with a constant value:

$$g(x) = \alpha f(x) + \beta \tag{3.5}$$

The parameters $\alpha > 0$ and β are the gain and bias parameters, which control the contrast and brightness respectively.

Figure 3.5: Image after dimming the light

f(x) is the source image pixels and g(x) is the output image pixels. Then, more conveniently the expression can be written as:

$$g(i,j) = \alpha \cdot f(i,j) + \beta \tag{3.6}$$

The second variation presents a combination of two detection functions. The main goal of this technic is to decrease the percentage of the wrong detections. After additional testing, it was noticeable that the HAAR face detection is more accurate than the other features; this can be due to the fewer features it is using.

3.2 Detailed Implementation

The following section will provide a detailed presentation for each step shown in Figure 3.1. The algorithm is implemented in C++ language using OpenCV and Dlib image libraries.

It also should be noted, that the algorithm objective is to detect and extract the face and not the facial landmarks.

3.2.1 Basic face detection algorithm

First, a module file is loaded for each algorithm. The model file basically contains the values and the data obtained after the training the feature.

Listing 3.1: Loading the classifier

```
1  //facedetection using HAAR
2 facedetection.load("../haarcascade_frontalface_alt2.xml");
3 //facedetection using LBP
4 facedetection.load("../lbpcascade_frontalface_improved.xml");
5 //facedetection using DNN
6 ConfigFile ="../dnn/face_detector/deploy.prototxt";
7 WeightFile ="../res10_300x300_ssd_iter_140000.caffemodel";
8 net=cv::dnn::readNetFromCaffe(ConfigFile,WeightFile);
9 //facedetection using HOG
10 hogFaceDetector = get_frontal_face_detector();
```

The DNN module used in this thesis is trained using a 300×300 image resolution as input. In the experiment section 4 the images are first resized to fulfill the 300×300 resolution condition.

After loading the files, the input images or frames are loaded and read using the OpenCV image library.

The VideoCapture can read a video from a given path or read the camera stream by setting the video path to 0.

Listing 3.2: HAAR and LBP detection funtion

```
1 //read frames from camera or video
2 VideoCapture cap(VideoPath);
3 cap >> frame;
4 //read image
5 Input = imread(ImagePath, cv::IMREAD_COLOR);
```

After loading the module files for each method, the main detection function is called. HAAR and LBP have the same detection function which has multiple parameters as shown in Section 3.1.

Listing 3.3: HAAR and LBP detection funtion

```
1 facedetection.detectMultiScale(Inputimage, Faces, ScaleFactor
2                                , minNeighbors, minSize, maxSize);
```

Concerning DNN, extra steps should be done before detecting. In the following code snippet, the image is converted over to a blob and sent through the system utilizing the forward() function. The yield detections is a 4-D network, where the third dimension iterates over the detected faces. (i is the iterator over the number of faces), the fourth dimension contains data about the bounding box and score for each face.

Listing 3.4: DNN detection funtion

```
1 inputBlob = cv::dnn::blobFromImage(InputData,inScaleFactor
2                                , cv::Size(), meanVal, false, false);
3 net.setInput(inputBlob, "data");
4 cv::Mat detection = net.forward("detection_out");
5 cv::Mat detectionMat(detection.size[2], detection.size[3]
6                      , CV_32F, detection.ptr<float>());
```

For instance, detections[0,0,0,2] gives the certainty score for the first face, and detections[0,0,0,3:6] give the bounding box and as long as the confidence or the certainty score is higher than the confidence threshold then the face will be detected.
The output coordinates of the bounding box are normalized between [0,1]. Along these lines, the directions ought to be increased by the height and width of the original picture to get the right bounding box on the picture.

Listing 3.5: DNN detection box

```
1 for (int i = 0; i < detectionMat.rows; i++)
2 {
3 float confidence = detectionMat.at<float>(i, 2);
4   if (confidence > confidenceThreshold)
5   {
6  int x1 =static_cast<int>(detectionMat.at<float>(i,3)*FWidth);
7  int y1 =static_cast<int>(detectionMat.at<float>(i,4)*FHeight);
8  int x2 =static_cast<int>(detectionMat.at<float>(i,5)*FWidth);
9  int y2 =static_cast<int>(detectionMat.at<float>(i,6)*FHeight);
10 ellipse(Inputimage, cv::Point((x2-x1)/2+x1, (y2 - y1)/2+y1),
11 ,(y2 - y1) / 2+y1), cv::Point((x2 - x1) / 2, (y2 - y1) / 2));
12    }
13 }
```

The last feature is HOG. The image should be first converted to a Dlib image considering that the original image was loaded using OpenCV Image library which makes it an OpenCV image.

Listing 3.6: HOG detection funtion

```
1 // Convert image format to Dlib's image format and detect
2 cv_image<bgr_pixel> DImg(InputImage);
3 vector<dlib::rectangle> faces = hogFaceDetector(DImg, SFactor);
```

3.2.2 Image Preprocessing implementation

Subsequent to loading the video or the picture, image preprocessing methods can be called before inserting the input data in the fundamental face detection function using the methods presented below.

The transformation of an image to a grayscaled image is done by an OpenCV method called cvtColor, which convert the image to a black and white image. This method can also convert the image to another image type like Lab, HLS...

Listing 3.7: Preprocessing methods

```
1 //Grayscale
2 cvtColor(Input, GrayscaledInput,cv::COLOR_BGR2GRAY)
3 //Resizing
4 cv::resize(Input , Res_Input, cv::Size(Width, Height));
5 //Filter and Denoising
6 GaussianBlut(Input, FilteredInput);
7 //Brightness
8 frame.convertTo(Res_image, -1, alpha, beta);
```

The resizing is also performed using OpenCV which needs the wanted new size of the image as a parameter. For the brightness method, other parameters are required which determine the α and β values.

3.2.3 Feature extraction combination

The added detection function is HAAR eye detection function. First, this function will be executed before the main face detection method (HAAR and DNN face detection) is used. After detecting all the eyes in the picture or video frame, the face detection function is called, and if the face contains the detected eyes then it will be detected, if not it will be neglected.

The functions are combined in the snippet below.

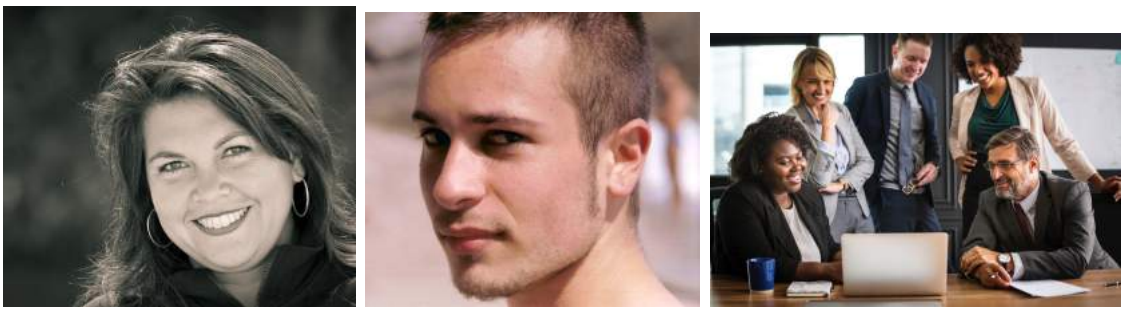
Listing 3.8: Combination of face and eye detecion

```
1 for (int j = 0; j < eyes.size(); j++)
2  {
3     {
4 if (faces[i].x < eyes[j].x && faces[i].y < eyes[j].y &&
5 faces[i].x + faces[i].width > eyes[j].x + eyes[j].width &&
6  faces[i].y + faces[i].height > eyes[j].y + eyes[j].height) {
7 ->Detection
8                     }
9           }
10 }
```

Chapter 4

Evaluation

To test the algorithms presented in the previous sections, three different data sets of pictures with a resolution higher than HD resolution are used: the first set contains frontal faces where the person in the picture is facing the camera directly, and the face regions (eyes, noise, etc..) are easily detectable. The second set contains non-frontal pictures, people are facing sideways, making some facial expression or the face is blocked by an object (hair, hand, etc.), and the third set contains multiple faces in one picture. Figure 4.1 shows examples from the different sets.
In addition to these sets, an extra set of images (Figure 4.2) is taken to be tested in the distance test Section 4.4. This set contains people standing at different distances to the camera.
The results will be then tested depending on accuracy, false positives and the computation time (Average detection time (ADT)), which also depends on the used PC.
The PC has an Intel(R) i7-7500U CPU @ 2.7 GHz to 2.9 GHz 8 GB RAM and an 8 GB with a 64-bit windows 10 operating system.



(a) Set one example (b) Set two example (c) Set three example

Figure 4.1: Different used sets

Figure 4.2: Extra set

4.1 Parameter test

OpenCV and DLIB modules have included parameters as mentioned in Section 3.1. The objective of this section is to tweak these parameters to get the best possible results.

4.1.1 Scale factor

All four features discussed in this thesis have the scale factor as a common parameter. The scale factor for HAAR and LBP is varied between 1.05 and 1.25 with a step of 0.05. For HOG the value will be 0, 0.1, 0.2, and 0.3 where 0 means no scaling, and for DNN, it will take 1.0, 1.1, and 1.2 as values, where 1 means no scaling.
A general inspection on HAAR result Table 4.1 and LPB result Table 4.2 shows that image scaling has almost the same effect on the results.

On the one hand, making the scale factor bigger damages the detection accuracy,

Table 4.1: HAAR scalefactor test.

		Scale factor				
Sets	**Tests**	1.05	1.1	1.15	1.2	1.25
Set 1	Accuracy	93 %	89 %	85 %	88%	81 %
	False positives	59 %	29 %	26 %	9%	13 %
	Average detection time	3.66	1.85	1.36	1.05	0.80
Set 2	Accuracy	86 %	78 %	78%	65%	57 %
	False positives	56 %	34 %	16%	13%	9 %
	Average detection time	2.55	1.32	0.98	0.76	0.58
Set 3	Accuracy	67 %	64 %	61 %	56 %	54 %
	False positives	54 %	29 %	18 %	16 %	6%
	Average detection time	2.29	1.19	0.88	0.68	0.52

especially in set two and three, where they contain non-frontal faces or a group of per-

sons. The reason behind this is that with a higher scale factor the chance of detecting smaller faces will decrease and thus they will be neglected.

On the other hand, it can be noticed, that the false positives are getting better and

Table 4.2: LBP scalefactor test.

		Scale factor				
Sets	**Tests**	1.05	1.1	1.15	1.2	1.25
Set 1	Accuracy	89 %	88 %	87 %	85%	80 %
	False positives	22 %	5 %	5 %	4%	3 %
	Average detection time	2.94	1.49	1.11	0.87	0.66
Set 2	Accuracy	82 %	75 %	60%	57%	45 %
	False positives	10%	4 %	0 %	2%	8 %
	Average detection time	1.75	0.91	0.68	0.54	0.41
Set 3	Accuracy	70 %	65 %	62 %	61 %	56%
	False positives	14%	6%	2%	2 %	2%
	Average detection time	1.28	0.65	0.50	0.38	0.30

the algorithm is detecting less wrong faces and needs less computation time. This is due to the fact that higher scale factor means that the algorithms compute the features on fewer levels of the picture .

The same approach can be done for HOG. The accuracy and the ADT results in Table 4.3 shows a different reaction to scaling comparing with the results in the two first tables. Scaling an image will produce multiple histograms that need to be calculated and combined and if a histogram detects a face pattern in one level and another histogram in another level do not, then that face will be discarded. The explanation to the declination of the computation time with the upscaling can be explained by the computation time dependency on the overall detected faces. Tanking set 2 as an example; the wrong detected face percentage went from 3% to 0% (scale factor 0.2 to 0.3) which means although the scale factor (0.2) in less than (0.3), the algorithm consumed

additional time detecting the 3% wrong faces.

Table 4.3: HOG scalefactor test.

		Scale factor			
Sets	**Tests**	0	0.1	0.2	0.3
Set 1	Accuracy	98 %	98 %	97 %	92%
	False positives	5 %	4%	3 %	2%
	Average detection time	1.94	2.18	2.13	2.01
Set 2	Accuracy	90 %	89 %	85%	85%
	False positives	3%	3 %	3 %	0%
	Average detection time	1.16	1.14	1.12	1.04
Set 3	Accuracy	80 %	78 %	74 %	72 %
	False positives	14%	06%	0%	0 %
	Average detection time	0.79	0.74	0.76	0.77

Table 4.4: DNN scalefactor test.

		Scale factor		
Sets	**Tests**	1	1.1	1.2
Set 1	Accuracy	97 %	97 %	97 %
	False positives	46 %	46%	47 %
	Average detection time	0.057	0.053	0.052
Set 2	Accuracy	96 %	96 %	96%
	False positives	38%	38 %	38 %
	Average detection time	0.053	0.053	0.053
Set 3	Accuracy	97 %	98 %	97 %
	False positives	3%	2%	3%
	Average detection time	0.053	0.054	0.055

Between all the discussed algorithm, DNN showed a really good ADT and accuracy percentage, which stayed almost constant in all three tests. The disadvantage of this method is that although the scale factor has been varied is that the percentage of the false positives in set 2 and set 3 did not get any better.

4.1.2 minNeighbors

In the previous section, the minNeighbor parameter was set on 3. Varying this variable may produce a more optimal result for HAAR and LBP keeping the ScaleFactor on 1.2 for HAAR and 1.15 for LBP.
Table 4.5 and table 4.6 shows the results for the both neighbor values 2 and 4. As for the scale factor test, a lower minNeighbor value has better accuracy, but worse false positives and detection time.

Table 4.5: HAAR minNeighbor test

		minNeighbor	
Sets	**Tests**	2	4
Set 1	Accuracy	84 %	79 %
	False positives	14 %	9 %
	Average detection time	1.13	0.98
Set 2	Accuracy	72%	59 %
	False positives	21 %	6 %
	Average detection time	0.9	0.76
Set 3	Accuracy	58%	52 %
	False positives	24%	10 %
	Average detection time	0.774	0.67

Taking set two in table 4.5 as an example and setting the minNeighbor to 2 improves the accuracy by 7%, from 65% to 72%. A major drawback is that the percentage of

Table 4.6: LBP minNeighbor test

		minNeighbor	
Sets	**Tests**	2	4
Set 1	Accuracy	88 %	84 %
	False positives	12 %	4 %
	Average detection time	1.15	1.11
Set 2	Accuracy	65 %	58 %
	False positives	11%	2 %
	Average detection time	0.672	0.671
Set 3	Accuracy	64 %	60 %
	False positives	7%	1%
	Average detection time	0.5	0.48

the false positive is worst now. The algorithm detects now 8% more wrong faces and almost 0.15s slower.
On the contrary, if the value is set higher, in this case, 4, we get a better detection time and false positive percentage, but the accuracy is aggravating.

So either the minNeighbor is set higher or lower, these values have their advantages and drawbacks. Three will still then be the minNeighbor parameter.

Comparing the output images of both features, it can be observed that HAAR detection box enclose the majority of the face (Figure 4.3(a)).
The LBP box is cutting off some parts of the eyes and mouth (Figure 4.3(b)).
This issue can be fixed by adjusting the detection parameter. Varying the size parameter in the ellipse function and changing it to

$$Size(faces[i]).width/1.9, faces.height/1.2)$$

will result a much better detection box (Figure 4.4).

(a) HAAR detection box (b) LBP detection box

Figure 4.3: HAAR vs LBP detection box

Figure 4.4: LBP detection box improvement

4.1.3 Confidence threshold

The confidence threshold set for DNN in the previous section was equal to 0.75. In this section, the confidence threshold will be lowered and raised to test the different outcomes of this feature.

As the following table lays out, lowering the confidence value gives a chance to the algorithm to detect more faces, in the third set it caught 100% of the faces which is 3% higher than when the scale factor was set to 0.75. Nevertheless, this confidence drop has also affected the wrong detection percentage.

Contrastingly, setting the confidence higher decreases both accuracy and false positives

Table 4.7: DNN Confidence threshold test

		Confidence Threshold		
Sets	**Tests**	0.65	0.85	0.95
Set 1	Accuracy	99 %	96 %	86 %
	False positives	50 %	41 %	37 %
	Average detection time	0.053	0.053	0.052
Set 2	Accuracy	98%	95%	93%
	False positives	40 %	38%	30 %
	Average detection time	0.053	0.053	0.054
Set 3	Accuracy	100 %	94 %	86 %
	False positives	3 %	2 %	0 %
	Average detection time	0.052	0.053	0.052

percentage, which still undoubtedly high, and with no perceivable ADT win.

4.1.4 Conclusion

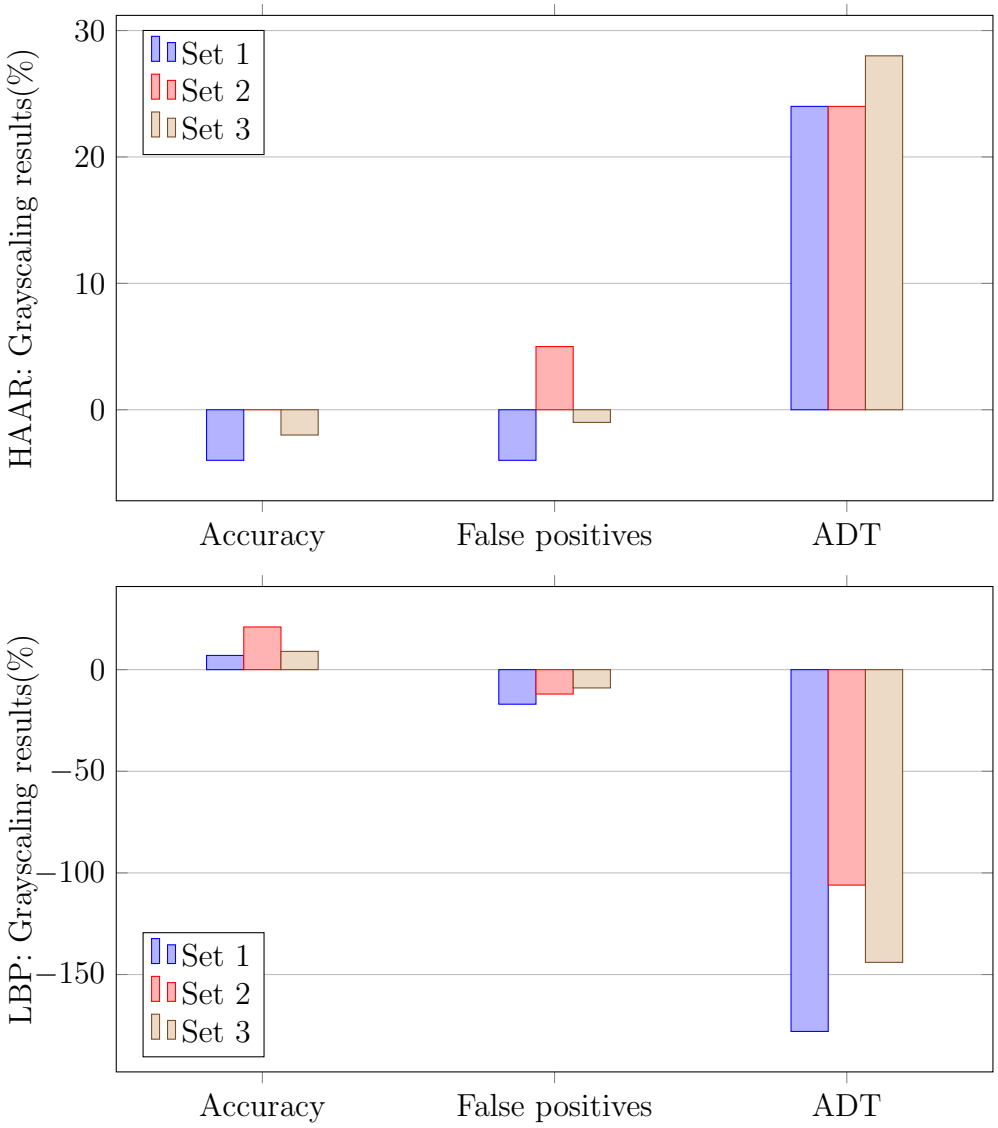
Changing the detection function parameters of the algorithms has delivered improvements especially for HAAR and LBP.

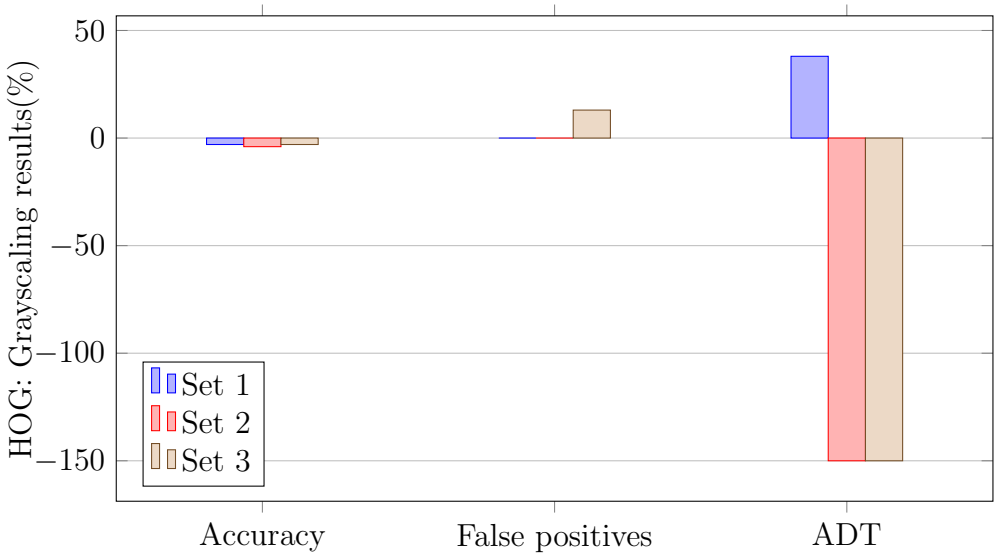
HOG and DNN did not win much from the variation of their scale factor which demonstrated that there is no need to vary the scale factor parameter, but they showed better accuracy results than HAAR and LBP. The major drawback in the parameter test is that DNN has a tremendous wrong detection rate and HOG computation time is the worst in comparison with the other methods.

4.2 Image preprocessing

The results showed in the previous section shows that every feature has its advantages and disadvantages. In this section, different image preprocessing technics will be put in test to boost up the weak points of the four features.

4.2.1 Grayscale





Grayscaling, as mentioned in this section's introduction, is transforming the frame from a three channel image (e.g RGB) to a single channel image. The only features that will be tested are HAAR, LBP, and HOG and that is because DNN accepts only three channel image.

The assumption is that grayscaling will boost the computation time seeing that the image will be transformed from a three channel image to a one channel one and the algorithm will spare almost 30% of the execution time.

The HAAR results prove these assumptions, for the three different sets, the ADT decreased by nearly 24% in comparison with the results chosen in the parameter test section. The downside of this technic is that the accuracy went down but only with 4% and 2% in set 1 and 3.

The reason behind this drop is that the features used to detect the faces as well as improving the accuracy are around three times less in a grayscaled image than a colorful one.

The dominant win of this preprocessing method is the ADT of HAAR feature as presented in the chart, although we added a task into the algorithm to convert the input data to grayscale. That is explained by the number of channels an input image has. The more channels an image has, the more features are being used, and the more time the system needs to execute all its steps.

LBP and HOG did not get any win by grayscaling the input image. For LBP the reason is apparent because this feature already does this transformation as a first step before passing it to the detection function.

4.2.2 Resolution test

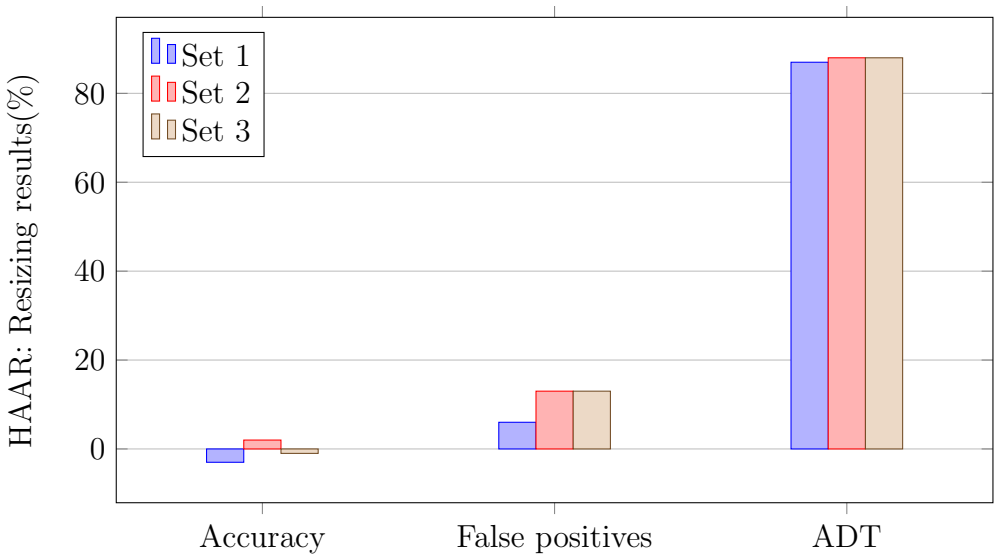


Figure 4.5: HAAR resizing results

Image resizing is one of the vital image preprocessing technics. For HAAR, LBP, and HOG, the Input picture are resized by dividing its original resolution by three which produces images with an almost HD resolution, For DNN the input pictures which are already a 300 by 300 pictures will be resized to 200 by 200.

The table 4.5, 4.6 and 4.7 returns excellent results concerning the false positives and the computation time which decreased seemingly with a minimum of 79%. It is clear that resizing an image decreases the number of pixels contained in an image and the features are executed faster in a smaller window.

The accuracy has been lightly damages using HAAR and HOG features, but it can be neglected seeing the win in the false positives and ADT.

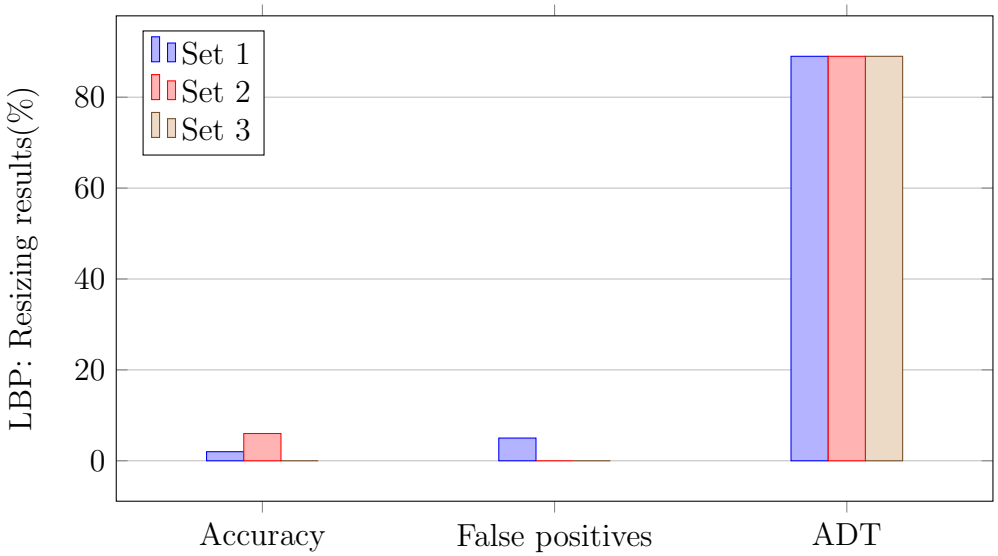


Figure 4.6: LBP resizing results

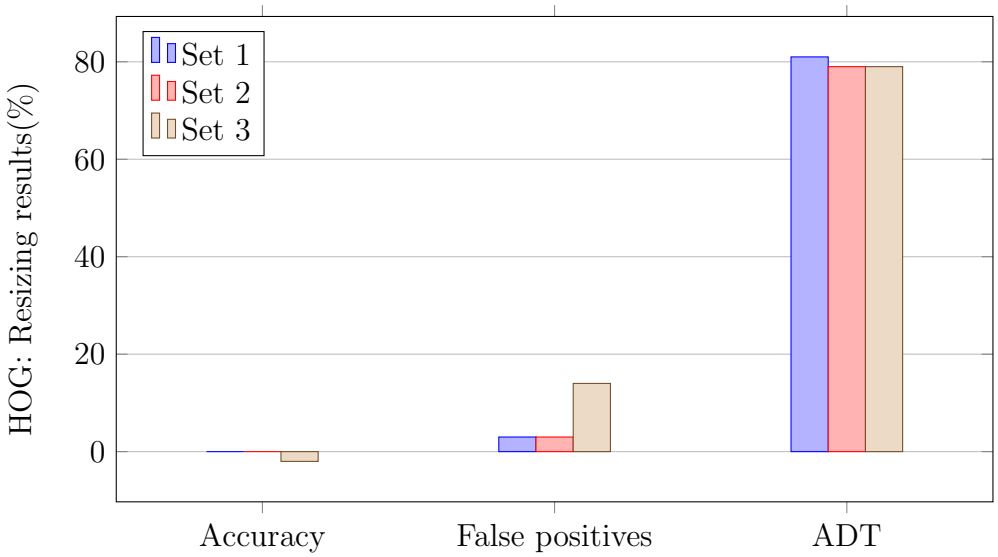


Figure 4.7: HOG resizing results

The same case can be noticed in Figure 4.8, the results in the parameter section were not too promising considering that the algorithm has detected too many wrong faces, but after the resizing, the percentage of false positives went from 46% to 3% in set one with a win of 43%, and it is needed to mention that DNN detected in set 1 100% of the excited faces.

It is also noticeable that the accuracy percentage has slightly decreased notably in set

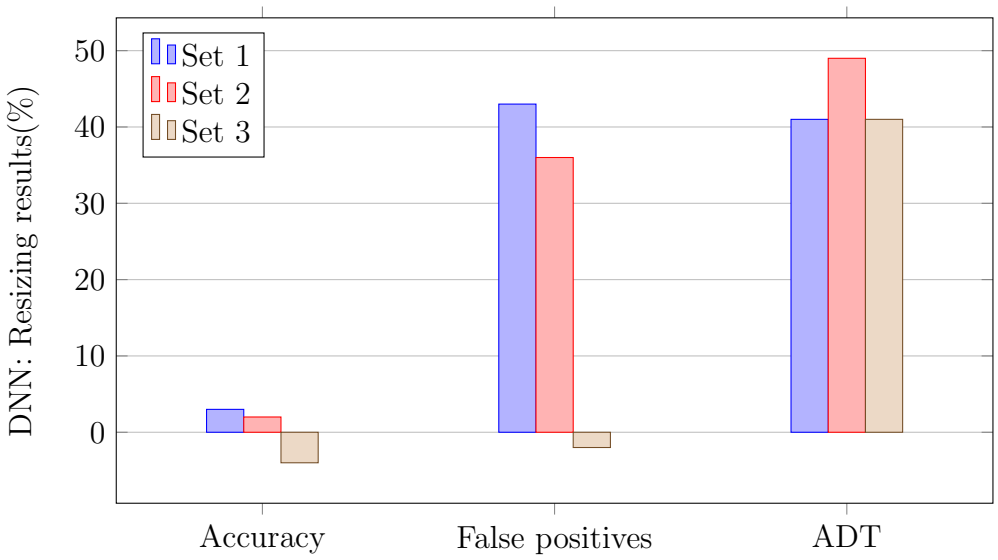


Figure 4.8: DNN resizing results

3 for all the features which can be interpreted as follow: the resizing process results in a smaller image which contains multiple faces located in different levels of the picture, the faces further from the camera will be too small to detect and thus the algorithms will not be able to identify them.

4.2.3 Brightness

In this section, the brightness of the images will be decreased knowing that in a conference room, the lightning can be high which as a consequence will make it harder for the algorithms to detect faces.

The brightness changing has improved the computation time of the HAAR features, but at the same time, it has affected the other test aspects negatively (Figure 4.9).

On the contrary, LBP needs a longer time, and with no visible improvement, it is safe to say that this method worsens the LBP potential (Figure 4.10).

This technic has no noticeable improvement when detecting with the HOG features. Although it has slightly boosted the ability of the system to detect less wrong faces, it did not bring much considering the accuracy. These results prove that the HOG feature is immune to the illumination changes.

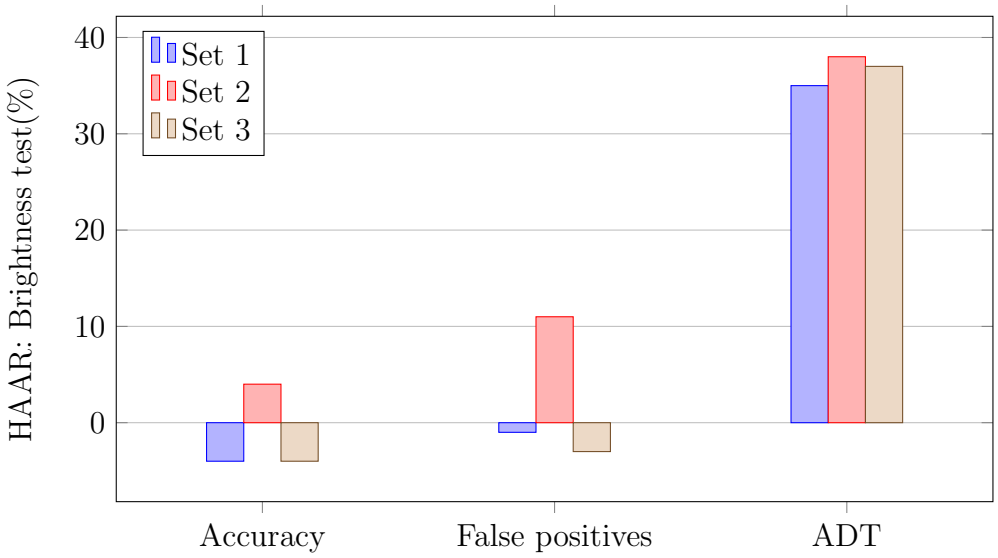


Figure 4.9: HAAR Brightness results

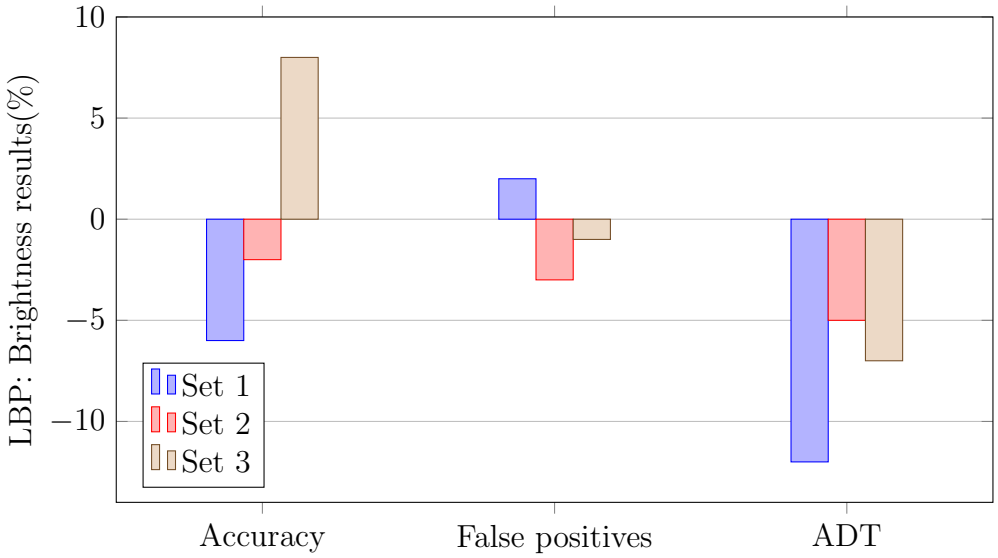


Figure 4.10: LBP Brightness results

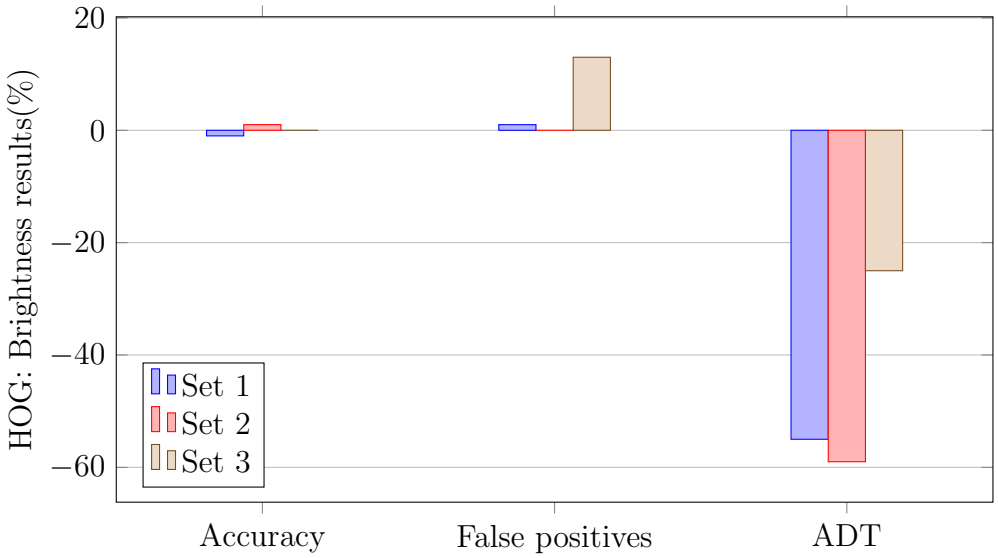


Figure 4.11: HOG Brightness results

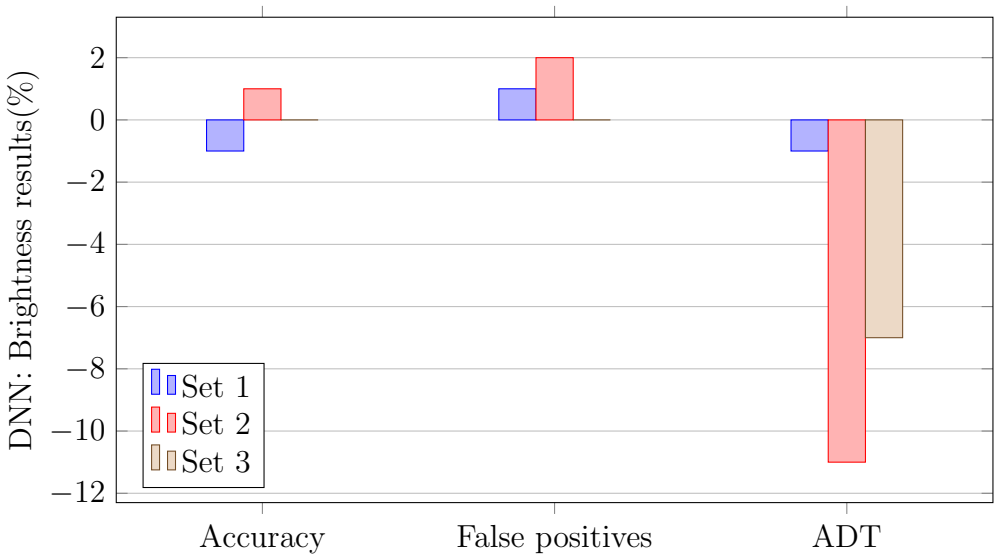


Figure 4.12: DNN Brightness results

As shown in Figure 4.12 DNN false positives percentage has slightly improved compared to the initial tests which were terrible, but it caused a loss in the accuracy and the ADT percentages.

Although this test did not bring any useful improvements, it is a must to mention that the pictures used for testing have a wide brightness variation and when the output pictures where investigated, it was easily seen that the darker images detected in the

previous test were not identified, but the brighter ones that were not detected in the tests above have been detected, so this technic can be helpful in an environment where the brightness is high and constant.

4.2.4 Filter and De-Noising

The following tables show the results for the four algorithms. The first three algorithms, HAAR, LBP, and HOG are made faster by this preprocessing technic.
The HAAR feature seems to lost a tiny accuracy percentage in the first set, and an insignificantly gain in the rest of the sets and the rate of the false positives.

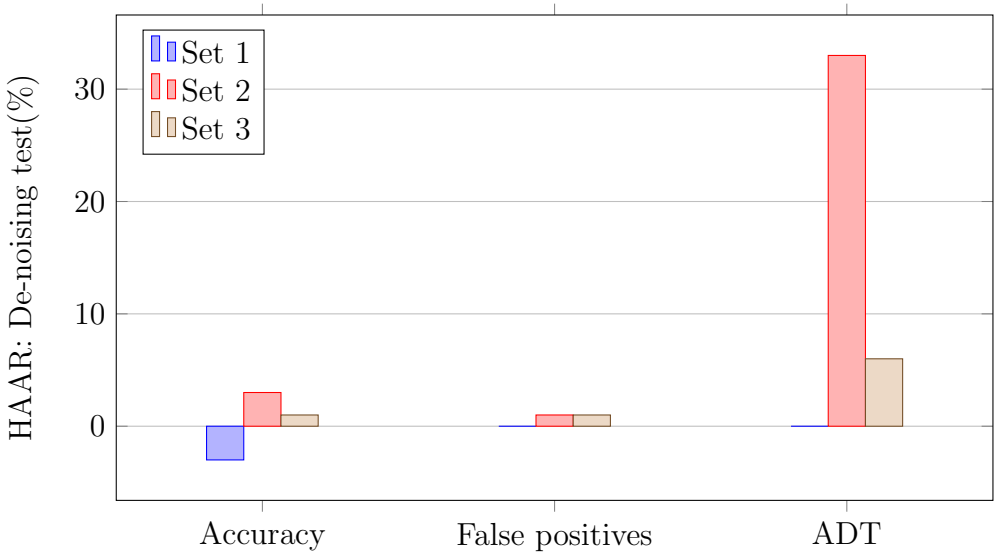


Figure 4.13: HAAR: Denoising results

The False positives number for LBP did get lower than for HAAR, but it did lose a seemingly amount of accuracy like shown in Table 4.14.
The primary winner of this preprocessing technic is HOG. In all testing results, the potential of this feature got better, which means that the image denoising produces a better image histogram and the calculation of this histogram is done in a smaller time. Even though the rest of the feature did improve in one aspect of the test or another, DNN did not benefit after image denoising.

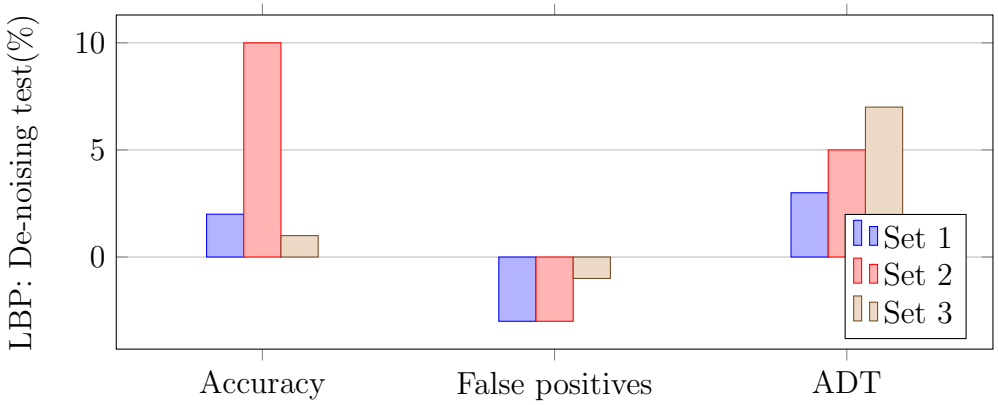


Figure 4.14: LBP: Denoising results

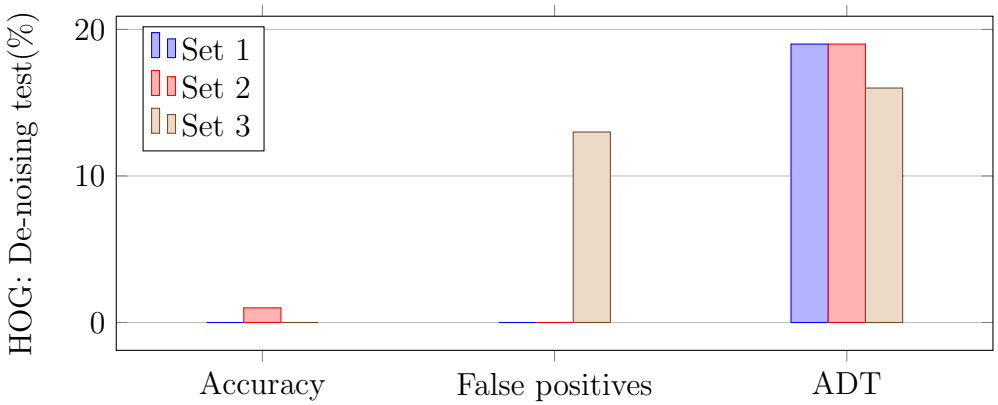


Figure 4.15: HOG: Denoising results

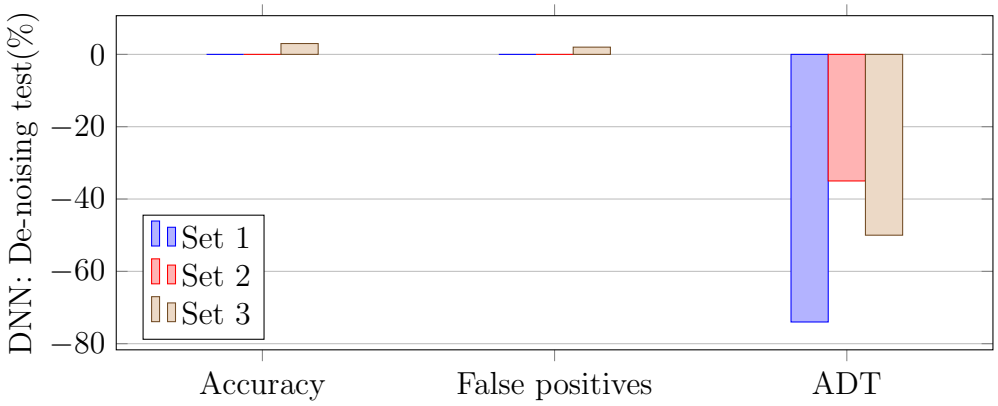


Figure 4.16: DNN: Denoising results

4.3 Algorithm combination

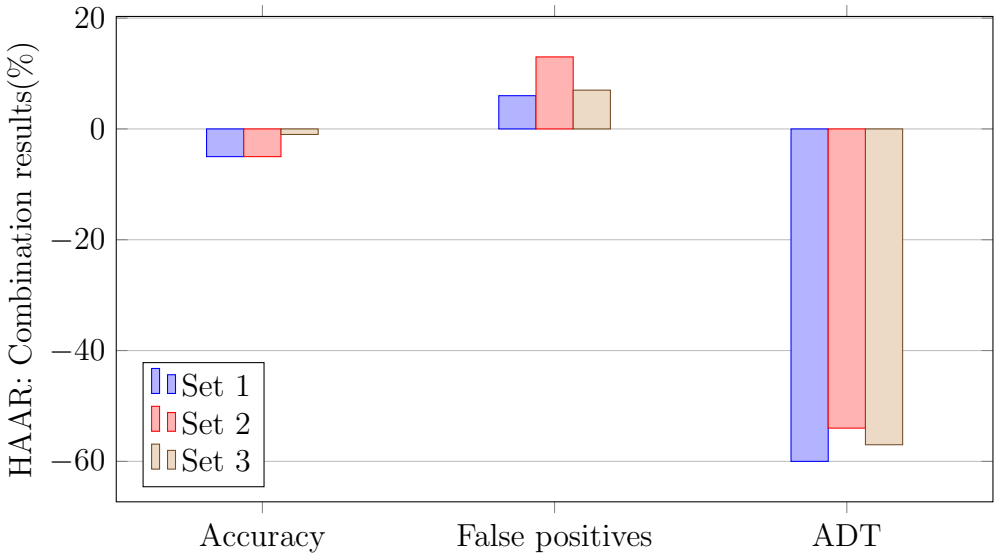


Figure 4.17: HAAR combination results

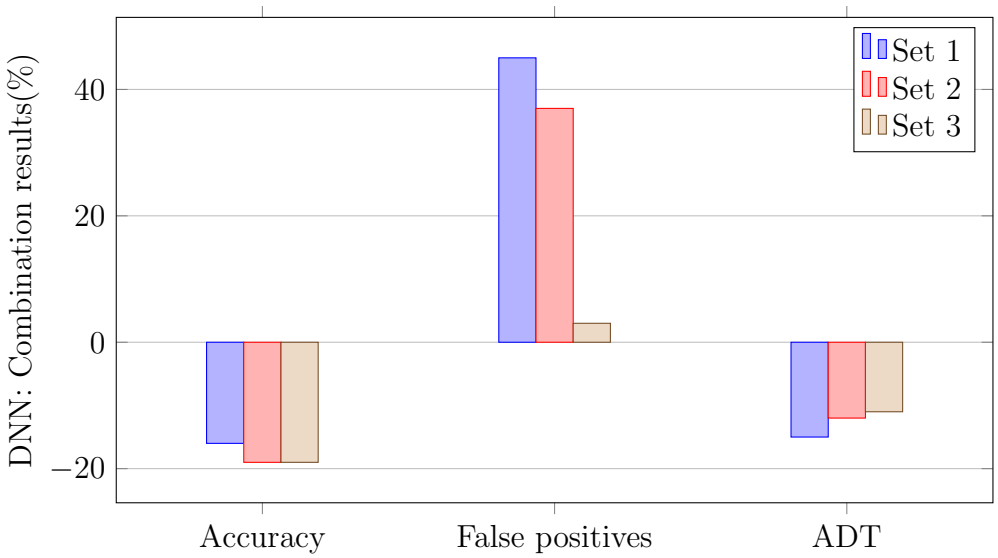


Figure 4.18: DNN combination results

In this section, both HAAR and DNN features are combined with an eye detection algorithm. Both algorithms seem to detect less false positives. The False positive percentage for DNN went from 46% to 1%. It is safe to say that this technic has

helped the algorithms to filter the wrong faces better. However, as for all the technics used in this thesis, this technic has a downside. The accuracy in both figures went down especially for the DNN feature and the ADT increased which is typical for the fact that in addition to detecting faces, the algorithm needs first to recognize the eyes.

4.4 Distance test

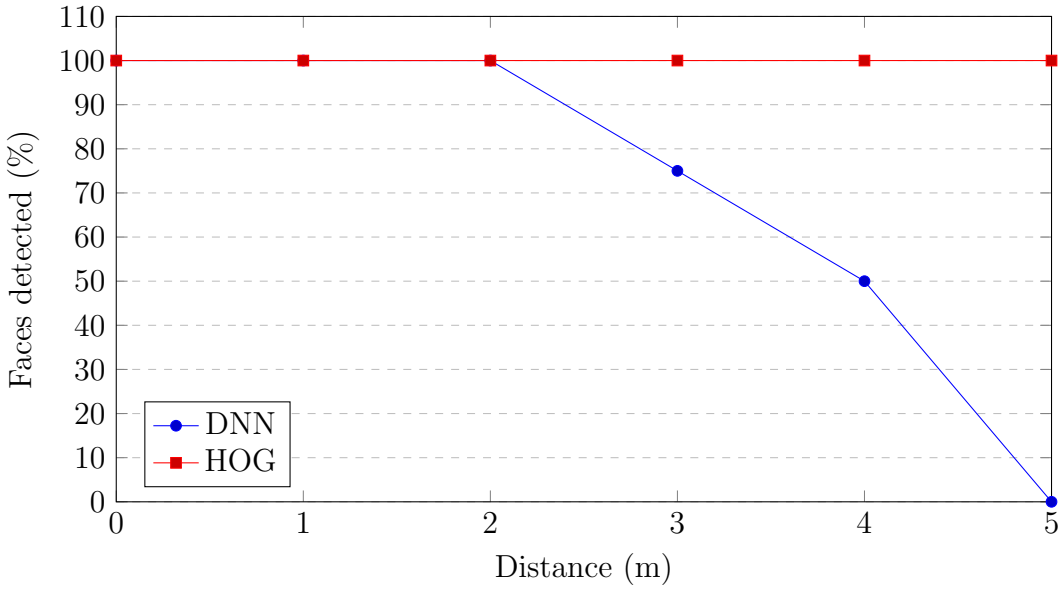


Figure 4.19: Distance test

In this section the only two tested algorithms are HOG and DNN, seeing that they showed the best results in the preprocessing section.

DNN resizing (Section 4.2.2) was the best preprocessing technic by far and knowing that the scale factor variation did not give any improvement, and the value for both of the algorithm stayed as it is (no scaling), a distance test is required to see how far a participant should be from the camera to be detected.

The above Figure 4.19 the distance test results.

DNN features start failing to detect the persons standing three meters away from the camera. At five meter it has not identified any face; on the other side, HOG showed a consistent performance detecting all the faces located near or far from the camera.

4.5 Framerate

Figure 4.20: DNN Frame rate

Figure 4.21: HOG Frame rate

In this section, a frame rate test was executed on both HOG and DNN. The Figures above show that DNN has the highest Frame rate per second (FPS) comparing to HOG. These results were expected, knowing that DNN had the lowest computation time in the past tests and that what makes it more suitable to be used to detect the faces in video stream thanks to its accuracy and rapidity of detection.

Chapter 5

Conclusion

In order to determine a reliable face detection technic to detect and extract faces in a teleconference environment, HAAR, LBP, DNN, and HOG were passed through different tests. This thesis demonstrated that these features could be improved using multiple technics and combination of different algorithms, which had variable influences on the improvement of the methods. The tests showed that the best features between the presented ones are DNN and HOG.

The DNN performance was perfect when resizing the input data. The results delivered best accuracy percentage and worked for different face orientations, up, down, left, right, side-face, etc. and even under substantial occlusion. Moreover, DNN works faster than the other features on CPU. The limitation of this feature is that the input frames should be resized in a way that the chances of detecting faces further from the camera are decreased.

HOG also provided excellent results; it works very well for frontal and slightly non-frontal faces and has Light-weight model as compared to the other three methods and does not get affected by the illumination changes. The drawback of HOG is that it does not work well on substantial occlusion but only under small occlusion and it works slightly slower than DNN.

To sum things up, it is safe to say that the best detection function, and after considering the different test aspects, is DNN. However, this statement can not deny that other

face detection technics may deliver similar or better result to DNN. Furthermore, the features handled in this thesis can be more improved by training them on a larger and better set of images and by combining them with other machine learning algorithms.

List of Figures

List of Tables

Bibliography

[Ada19] *AdaBoost*. https://en.wikipedia.org/wiki/AdaBoost, Mar. 2019

[DLI19] *DLIB*. https://en.wikipedia.org/wiki/Dlib, Mar. 2019

[DT05] Dalal, N. ; Triggs, B.: Histograms of oriented gradients for human detection. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* (2005), Ju.

[Gau18] *Gaussian blur*. https://en.wikipedia.org/wiki/Gaussian_blur, Nov. 2018

[HOG16] *Histogram of Oriented Gradients*. https://www.learnopencv.com/histogram-of-oriented-gradients/, Dec. 2016

[Kin09a] King, Davis E.: Dlib-ml: A Machine Learning Toolkit. In: *Journal of Machine Learning Research* (2009)

[Kin09b] King, Davis E.: A Survey of Face Recognition Techniques. In: *Journal of Information Processing Systems* 5 (2009), Ju., Nr. 2, S. 41–68

[Ope19] *OpenCV*. https://opencv.org/about/, 2019

[Pra17] Prado, Kelvin S.: *Face Recognition: Understanding LBPH Algorithm*. https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b, Nov. 2017

[TOH94] T. Ojala, M. P. ; Harwood, D.: Performance evaluation of texture measures with classification based on Kullback discrimination of distributions. In: *Proceedings of the 12th IAPR International Conference on Pattern* (1994), Aug.

[TOM02] T. Ojala, M. P. ; Maenpaa, T.: Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2002), Aug.

[VJ11] Viola, P. ; Jones, M.: Rapid object detection using a boosted cascade of simple features. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001* (2011), Dec.