university of groningen

faculty of science and engineering

Counting people in videos

Bachelor thesis

June 27, 2018

Student: Twan Schoonen

Primary supervisor: André Sobiecki

Secondary supervisor: Alexandru C. Telea

Abstract

In a more and more media filled world, automation is needed for analyzing media sources. In this thesis we focus on counting people in video collections. These videos collection are to large to watch manually and have a low resolution. We compare 2 different face detection techniques and 4 different image comparison techniques. With the optimal setting we achieve a reduction rate of 95.36% on average.

Contents

Chapter 1

Introduction

Video surveillance methods become increasingly widespread and popular within many organizations, including law enforcement, traffic control, but also residential applications. In particular, the police performs investigations based on searching specific people in videos and in pictures. As the number of such videos is constantly increasing, manual examination of all frames becomes impossible. Some degree of automation is strongly needed.

Lets us start with an example, in China there are already smart surveillance cameras used on daily basis. These cameras do not only just record videos but there is actually object detection being applied as is seen in Figure 1.1. In this Figure we see that the objects attributes are also recognized by these cameras. So a car color and model, and even if a person wears long or short sleeves. This means you can ask these cameras complicated queries like, was there a man walking by at 12 am? Even more interesting, give a signal if you see person X. That being said in China alone there are already 176 million of these surveillance cameras [3]. Consider that actual people would have to manual watch through these days of footage.

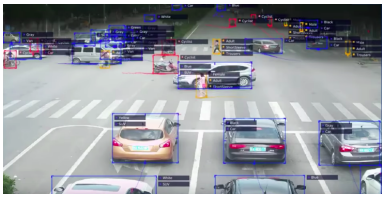


Figure 1.1: A Chinese smart surveillance camera.

The goal of this project is to support a number of the aforementioned automation tasks. Specifically, given a large collection of videos, there are three main question that this thesis tries to find the answer to. We will introduce these questions in order and give a more detailed explanation below.

Question 1: How many different people are present in the video collection?

This is the most important question, and hence the main focus of this thesis. We do however only consider a person's face, and thus work with face detection. This means motions detection methods are not used, or other object detection methods. The reason that we only consider a face is due to the fact that recognizing a person only from his body is very hard, while searching for people's face is more acceptable. Once we are able to find the answer to this question a lot of video surveillance work can be automated, in a way that if there are no people present we do not have to watch the video manual again. And thus police, or other people working with videos, can now only watch relevant frames.

Let us think of an other place where this is a logical question to ask, amusement parks. We could inform visitors of these parks what areas are crowded and where less people are. Doing this results in a win-win situation, for the park, we have a better distribution of people and hence the whole park is being used. For the customers this is beneficial, because in there experience it is less busy, thus less waiting times. Now assume that we now have this large collection of videos, where we already not have to watch a big chunk of the footage anymore, because no one was present. However we now want to search for a specific person. We would still have to manually watch all the surveillance material, to see when this person is present. This introduces another path where we can automate the work of policemen.

Question 2: How to group faces (present in the individual frames) per person, so that we can tell where and in which specific frames the same person appears?

Answering this question implies that we can ask a large video collection, give me all frames where person X is present. There are a lot of challenges for this to work, and finding a general method that can do this a 100% accurate is never done before. One of the challenges is aging. Lets assume we have recognized a person on particular day and this person commits a crime. The desired result then would be if we ever see him again to let the police know. However lets say 5 years later this person can look a lot different. Now lets say we have this automation working, in case of a crime someone would manually have to go though the (relevant) frames to see if they can find the frames best displaying the face of an criminal. An example of these manual selected frames in Figure 1.2. These manual selected frames are used to ask the public if they have seen this person.


Figure 1.2: Manual frame selections

In this Figure we see that two frames of some criminals. These frames are manually selected. Here an other problem with the possibility of automation is found. We will again mainly focus on faces here so the resulting images of the criminals will be just facial images. Doing this work manually is a task that is work intensive since analyzing frame by frame costs time. This brings us to research question 3.

Question 3: How to select a good "representative face" for the above groups? A representative face is one of good pose (orientation), resolution, lighting, contrast, and image sharpness.

Solving this question would bring an other great aspect for automation. These representative faces do not have to be selected manually. Analyzing all results form a modern day surveillance camera systems is virtually impossible. Since these groups are already clustered in some similar way we now need to represent these groups with a representational face.

So ideally we would be able to feed a media stream to our program, and this program then analyses the data, counting the persons present, displaying the most relevant faces, and save the detected persons to a global database. This can system can have a widespread of applications. We can think of a system monitoring parking spaces, or boarder control that is fully automated, but mostly automating the work for crime related videos.

The system that we will try to develop will not be 100% accurate since there exist no such method. We will however try to get the best results possible. In our results the false positives are acceptable, since we will just detect more people then present. Not detecting a person is worse since this means some people will come through our detection system and thus maybe walk freely. All automation in this field means less work, so if we can accurately cut the number facial images, then this would already help.

The project is done in cooperation with ZiuZ [4], a company interested in exploring these ideas for actual future usage in video surveillance projects. In this bachelor project, we present a method for summarizing faces from videos, many of these face images have poor resolution, we uses two types of face detection and then compare these faces by different type of histogram comparison techniques.

The structure of the thesis is as follows. In Section 2 we discuss the relevant work to answer these three questions. After discussing several methods the selected and implemented methods are more explained in Section 3. The results of these different methods will be presented in Section 4. And then we will compare these results to come to our conclusions in Section 5.

Chapter 2

Related Work

In order to find the best methods for counting people in videos, a literature study is performed. An ideal result for this project will be a software system able to:

- Read a video and detect and crop all the faces present in a frontal position.
- Count how many people are in the video as accurate as possible.
- Group faces based on similarity; for each group, select the best "representative" face based on quality criteria including resolution, sharpness, contrast, and lighting.

To obtain this ideal result multiple papers are discussed, on multiple topics. These papers are about our four steps of recognizing people in videos. For each of these four steps a more detailed description is given in the relevant section. In these sections we talk about relevant papers that fall in the discussed topic. At the end of this chapter a summary is given of some methods that we discuss and a small conclusion on our findings is provided. The upcoming sections are:

1. Face Detection 2.1, detecting the faces in each frame. After that the faces are stored in a database or in memory.
2. Image Comparison 2.2, since we save the faces for each frame we need some method to distinguish the frames that contain the same person, here image comparison comes to help.
3. Face Clustering 2.3, this process clusters different persons for example on race, gender, and age range. This then can be used for to ask complicated questions to the database. However we face the risk of losing identities by grouping them.
4. Face Recognition 2.4, this is the process of actually giving a label to each face, so instead of saying this is a face, we are able to say this face belongs to person X.

2.1 Face Detection

There are multiple ways to count people in videos. One way of doing this would be to look for shapes or motion. Another way would be to detect the faces. In this thesis we focus on the second option, because persons are more recognizable from their faces then from their body's. Detecting shapes or body parts, can end up with a database filled with people backs not being of any use.

The process of detecting faces is called face detection and for humans is task is quite easy. Explaining this to a computer is a more complicated task. Face detection can be seen as a specific case of object detection. In object detection the goal is to find all the locations and sizes of all objects in an image [1]. In face detection we only try to find one object type being the face of a human. A somewhat desired result is shown in Figure 2.1. Here we see that the green rectangles show where the faces are in this picture. Of course one picture can also been seen as an individual frame. Factors that make it difficult to come up with a general robust solution are:

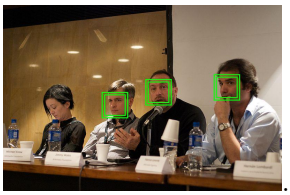


Figure 2.1: Face detection example

- Head rotation and tilt
- Lightning intensity and angle.
- Facial expressions.
- Aging.
- Image resolution
- Occluding objects

To overcome these problems multiple methods are developed of the last years. Each with their own advantages and disadvantages The most optimal result would hence be, loading a video finding all the faces that are present and saving these faces in a database. Again doing this 100% accurate is hard, and having no false positives rarely happens. Once a face detected, the face can be cut from the corresponding frame and resized to a desired output width and height. Detecting faces is widely used in combination with other techniques. In our project we use face detection in combination with image comparison. However there are direct applications of face detection. In photography, we use face detection to focus our lens to this area. This technique is already available on mobile phones. Snapchat [9] is another interesting example. Looking at Figure 2.2 we see one of Snapchat's many available filters. Here not only the face is detected but also the angle and then correctly drawing some overlay.

Figure 2.2: A Snapchat filter

This is quite remarkable if we take into account that Snapchat does this in real time on a mobile phone.

There are multiple categories for facial detection algorithms. We will list these categories below and discuss relevant papers.

2.1.1 Traditional Face Detection

Traditional face detection algorithms identify facial features by extracting landmarks, or features, from an image. One of the most used methods is proposed by Voila et al.[21]. Here multiple face features are combined to detect a face and thus this papar falls into this category. This method was a break though, because it was the first to detect faces in real time. The algorithm is fast while obtaining high detection rates, it has four main stages.

1. Haar Feature Selection
2. Creating an Integral Image
3. Adaboost training
4. Cascading Classifiers

The haar features are using the fact that all faces share some similar properties. For example the nose is a bit brighter that the two areas next to it, see Figure 2.3. Then the best haar features are combined. By doing this the speed of execution goes up. These best haar features are applied to all smaller windows of the current frame The downside of this algorithm is that is struggles with face angles in any direction (one of the problems stated). This is due to the fact that it is trained with frontal faces. An upside is that this method is already available in the OpenCV library [2] with trained cascades.

Figure 2.3: An example haar feature

2.1.2 Skin Texture Analysis Face Detection

DU Cui-huan, et al.[8] uses AdaBoost in combination with a skin model, and hence falls into this category. This method is applied to videos, and thus enhancing the relevance of the paper to the main research questions. Two phases are discussed, first the motion area is calculated. On this area the AdaBoost algorithm from Viola et al.[21] is used. Once this detection is successful clustering is done by a skin model. They obtain an accuracy of 92.3% on a database of 3000 faces.

2.1.3 Motion Based Face Detection

In Paul, et al.[18] a more general overview is given. Here it is not only focused on the detection of face but also people in general. Also it talks specifically about surveillance videos and hence is relevant. All methods are grouped in three categories: background subtraction, optical flow and spatio-temporal filters. These methods are used for detecting motion. Once motion is detected we can verify that it is a human by three major types: shape-based, motion-based and texture based. In this paper Table 2.1 is presented.

Methods	Accuracy	Computational time	Comments
Shape-based	Moderate	Low	Simple pattern-matching approach can be applied with appropriate templates. It does not work well in dynamic situations and is unable to determine internal movements well
Motion-based	Moderate	High	Does not require predefined pattern templates but struggles to identify a non-moving human
Texture-based	High	High	Provides improved quality with the expense of additional computation time

Table 2.1: Compaison of object classification methods in terms of accuracy and computational time [18]

However since our research is focused on the human face, this paper and DU Cui-huan, et al.[8] are not relevant to our proposed solution.

2.1.4 Deep Learning Face Detection

An other approach is suggested by Farfade et al.[10]. Here deep learning is used to detect faces and hence it falls into this category. All current benchmarks are done with deep neural nets, or more specifically convolutional neural nets. In this paper the problem of multi-view face detection is discussed. While there has been significant research on this problem, current state-of-the-art approaches for this task require annotation of facial landmarks. They also require training dozens of models to fully capture faces in all orientations. In this paper Deep Dense Face Detector (DDFD) is proposed, a method that does not require pose/landmark annotation and is able to detect faces in a wide range of orientations using a single model based on deep convolutional neural networks. The proposed method has minimal complexity; unlike other recent deep learning object detection methods, it does not require additional components such as segmentation, bounding-box regression, or SVM classifiers. Evaluations on popular face detection benchmark datasets show that their single-model face detector algorithm has similar or better performance compared to the previous methods, which are more complex and require annotations of

either different poses or facial landmarks.

One of the advantages about this paper is that the newest OpenCV [2] comes with functionality for this method.

2.1.5 High Tech Camera Face Detection

High tech cameras have more functionality, for example the camera is able to detect temperature or depth. This extra data could lead to even better face detection methods. We do not have relevant literature in this category. In out problem we do not have these high tech cameras.

After this literature research about face detection, we decide to implement both the Haar cascade from Voila et al.[21] and the method from Farfade et al.[10]. These methods will be compared. These are both available in the newest version of the OpenCV library and thus we will be using this. Implement one of these methods from scratch would not fit in the time given for a bachelors project. Once we present our results of the comparison we will choose one of these methods for our final people count method.

2.2 Image Comparison

Now that we discussed how we can detect faces and save these for each frame, we want to have a method for comparing if a face is of the same person or if it is someone different. Image comparison is a big theme and there are multiple methods discussed in this chapter. Methods of similarity between two images are useful for the comparison of algorithms devoted to noise reduction, image matching, image coding and restoration. In our thesis it is mainly focused on image matching. Facial images are a very high-dimensional feature vector. Usually in modern applications some way of reducing this dimensionality is used when working with faces. Most papers make a comparison between different image comparison techniques.

The in Gesù et al.[7] described methods based on distance functions. Measures of similarity are also used to evaluate lossy compression algorithms and to define pictorial indices in image content based retrieval methods. In this paper they develop a distance-based approach to image similarity evaluation and they present several image distances which are based on low level features. The sensitivity and effectiveness are tested on real data. Experimental results indicate better sensitivity of the functions by combining both global intensity and local structural features with respect to conventional intensity-based measures.

The next paper describes an empirical comparison Stevens et al.[20]. Using 27 different error functions on a set of 20 example problems. Each error function measures the similarity between an observed and a predicted image. The goal of the paper is to select an error best suited to guide a heuristic search algorithm through a space of possible scene configurations.

Another approach is given in Jia et al.[12]. In this paper, three newly proposed histogram-based methods are compared with other three popular methods, including conventional histogram intersection (HI) method, Wong and Cheung's merged palette histogram matching (MPHM) method, and Gevers' colour ratio gradient (CRG) method. They test their methods on vehicle license plates to classify them. Experimental results disclose that, the CRG method is the best choice in terms of speed, and the GWHI method can give the best classification results. Overall, the CECH method produces the best performance when both speed and classification performance are concerned.

As seen in the above mentioned papers, there are a lot of different methods for image comparison. All papers make a comparison between different methods. We however will also compare the in OpenCV [2] given histogram comparison methods. These methods are explained in Section 3.

2.3 Face Clustering

Now that we are able to find the faces from a video, and know what face is the same, we are left with a large database with faces. To further analyze all these faces, we need somehow a way to simplify the dataset, i.e. reduce it to a small one. This is typically done by putting very similar data items (faces) together, which in turn can be done by what are known as clustering methods.

Essentially, any clustering algorithm is an optimization process which receives 2 inputs: a distance function that compares data items, and a desired simplification level (expressed either in the maximal distance that 2 elements in the same cluster can have, or by the max size a cluster can have). Then, it partitions the data items into clusters. Of course, there's a trade-off: if you have fewer clusters, you'll have more items per cluster (more generalization or simplification, but higher error); and if you have more clusters, you'll have fewer data items per cluster (less generalization, but less error). Being able to cluster these faces will increase the functionality of our database. As an example use, if we want to only look at all the woman we detected. Then the faces should be clustered on gender.

In Otto et al.[16] the best practices for efficient clustering of face images are explored. They apply their clustering on a database of up to one million faces. This data set is then clustered in a large set of approximately 200 thousand clusters. The main challenges for clustering face images are:

- Large dataset size (millions of face images).
- Large number of classes: a crowd may contain a large number of individuals (tends of thousands, if not more).
- High intra-class variability and small inter-class sepa-ration: images are captured under unconstrained conditions, with uncooperative subjects, in difficult imaging environments.
- Unknown number of clusters: the number of individuals present in the collected data is not known a priori, but may contain tens of thousands of clusters.
- Variable number of samples per cluster: some individuals may be present in only a few images or video frames, others in many.

In the conclusion it is stated that, Rank-Ordered clustering is a good trade off between computational power and accuracy.

A paper discussing the given example about gender clustering is Orozco et al.[15]. It is stated that the good results obtained using deep convolutional neural networks in vision tasks make them an attractive tool to improve the capacities of gender recognition systems. They propose a deep convolutional network architecture to classify as male or female person. Haar features embedded in an AdaBoost are used to obtain candidate regions. The used data set is the common Labeled Faces in the Wildand Gallagher's dataset. Their results on the proposed architecture are 95.42% and 91.48% accuracy for the training set and for the test set.

In Otto et al.[17] a method for clustering a large collection of unlabeled faces is given. They address the problem of clustering faces into an unknown number of identities and thus unknown amount of clusters. They state that the problem applies to social media, law enforcement and other applications, where there can be hundreds of millions of faces present. An approximate Rank-Order clustering is presented that preforms better then other popular clustering algorithms. In their experiments there are up to 123 million faces that are clustered over 10 million clusters. Clustering results are analyzed in terms of external (known face labels) and internal (unknown face labels) quality measures, and run-time. Their algorithm achieves an F-measure of 0.87 on the LFW benchmark (13 K faces of 5,749 individuals), which drops to 0.27 on the largest dataset considered (13 K faces in LFW + 123M distractor images).

Face clustering is not the best choice for our project. This is because we do not want to lose any identity. This means at most one cluster per person which makes clustering not very useful. Another reason is that face clustering is complex and not doable in the given three months. As last face clustering is not possible in real time and there are not test of images with low resolutions. These reasons lead us to the conclusion that we do not implement or use any of the above given face clustering techniques.

2.4 Face Recognition

We discussed how to identify, compare and cluster faces. This means only the last step is left undiscussed, being able to recognize people once a face is detected. A facial recognition system is a technology capable of identifying or verifying a person from a digital image or a video frame from a video source. There are multiples methods in which facial recognition systems work, but in general, they work by comparing selected facial features from given image with faces within a database. In this field there has been a lot of recent advances. As an example usage, the Face ID introduced by Apple makes sure that the owner of the phone can log in using his or her face. Here first face detection is used to see where in the camera the face is present, hereafter face recognition is used to see that it is indeed the owner of the phone. To do this securely Apple needs very accurate recognition method, otherwise anyone can log into your phone. This technology even learns from changes in a user's appearance, and therefore works with hats, scarves, glasses and many sunglasses, beard and makeup. The advantage that Apple has is that the faces used to log in to a phone are mostly full frontal and of high resolution, or it just does not work. For our research we want to detect all faces and these faces are of low resolution so counting them complicates the process.

A paper discussing the current state-of-the-art is Schroff et al.[19]. In this paper a system, called FaceNet, that directly learns a mapping from face images to a compact Euclidean space where distances directly correspond to a measure of face similarity, is presented. Once this space has been produced, tasks such as face recognition, verification and clustering can be easily implemented using standard techniques with FaceNet embeddings as feature vectors. Their method uses a deep convolutional network trained to directly optimize the embedding itself, rather than an intermediate bottleneck layer as in previous deep learning approaches. To train this network, they use triplets of roughly aligned matching / non-matching face patches generated using a novel online triplet mining method. The benefit of this approach is much greater representational efficiency. The results are achieved with only 128-bytes per face. On the widely used Labeled Faces in the Wild (LFW) dataset, the system achieves a new record accuracy of 99.63%. On YouTube Faces DB it achieves 95.12%. This system cuts the error rate in comparison to the best published result by 30% on both datasets.

Another widely used method is discussed in Lei et al.[14]. In this paper, a method is proposed that is simple and efficient. It uses face representation feature that adopts the eigenfaces of Local Binary Pattern (LBP) space, referred to as the LBP eigenfaces, for robust face recognition. In the experiments, the proposed LBP eigenfaces are integrated into two types of classification methods, Nearest Neighbor (NN) and Collaborative Representation-based Classification (CRC). Experimental results indicate that the classification with the LBP eigenfaces outperforms that with the original eigenfaces and LBP histogram.

Hmani et al.[11] is interested in the reproducibility of face recognition systems. By reproducibility they mean: is the scientific community, and are the researchers from different sides, capable of reproducing the last published results by a big company, that has at its disposal huge computational power and huge proprietary databases? With the constant advancements in GPU computation power and availability of open-source software, the reproducibility of published results should not be a problem. But, if architectures of the systems are private and databases are proprietary, the reproducibility of published results can not be easily attained. To tackle this problem, they focus on training and evaluation of face recognition systems on publicly available data and software. Also they are interested in comparing the best Deep Neural Net (DNN) based results with a baseline "classical" system. This paper exploits the OpenFace open-source system to generate a deep convolutional neural network model using publicly available datasets. It is studied what the impact of the size of the datasets, their quality is and they compare the performance to a classical face recognition approach. The focus is to have a fully reproducible model. To this end, they used publicly available datasets (FRGC, MS-celeb-1M, MOBIO, LFW), as well publicly available software (OpenFace) to train our model in order to do face recognition. Our best trained model achieves 97.52% accuracy on the Labelled in the Wild dataset (LFW) dataset which is lower than Google's best reported results of 99.96% but slightly better than FaceBook's reported result of 97.35%.

This is a quite interesting point, since this means that it should be possible for researchers to also reproduce these results. However face recognition does not work for face images with poor resolution and they requires the face in standard position, light and distance. There are methods for face recognition using face images with poor resolution [13], but understanding and implementing these methods are not doable for the given three months of the bachelor project and the risk of not-success is high, that is the reason why we use histograms.

2.5 Summary

Below the Table 2.2 summarizing all relevant literature discussed in this chapter, is summarized. We consider the following four criteria for evaluating the methods in the

papers, validation, availability, complexity and usability. The reason for this is that we want next to pick an "optimal" method to build upon. Optimal here means that it satisfies as much criteria as possible. A more detailed description of what the criteria are is given below.

- Validation: How well the algorithm is tested by the authors in the paper, and how promising the results are looking for this project;
- Availability: Whether the source code is available on the Internet, or if not, whether the algorithm is described in enough detail for us to reproduce it;
- Complexity: Whether the algorithm could be executed fast enough to be close to real time;
- Usability: Refers to the number of parameters and their meaningfulness. A large number of parameters is bad for ease of use, and it should be easily understandable what a parameter affects and what the result of adjusting it will be.

Subject	Technique	Validation	Availability	Complexity	Usability
Face Detection	Haar cascade	+/- (struggles with angles)	++ (lot of sources online)	+	+ 3 parameters that all have default values
	Deep neural net	+ (Good detction problems: low resolution and far away)	- (hard to find examples and code)	++	+/- at least 5 parameters
	Motion based	+/- (No faces detected)	+ (lots of different solutions)	-	++
Image Comparison	Distance functions	+	– (No source code found)	+	+
	Empirical comparison	+	- (Not available)	+	+
	Histogram methods	+/- (Hard to have robust comparison with histograms)	++ (Available in the OpenCV library)	+ (can be done real time)	+ (Only a few parameters)
Face Clustering	Rank-ordered	++ (Good results on large datasets)	+/- (Found some suggestions not in OpenCV)	+/- (Can take time)	+/-
	CNN + Adaboost	+	– (No where found, not suggested)	+ (Fast after training)	- (Layers require settings)
Face Recognition	FaceNet	++ (99.63% state-of-the-art)	- (Not available in the OpenCV library	++ (Fast after training)	++ (Few parameters)
	LBP eigenfaces	+	+ (examples online, not in the OpenCV library)	+ (Reduces the vector space)	+ (Almost the same as histograms)

Table 2.2: Comparison of different relevant techniques

2.6 Conclusion Of The Relevant Work

As stated in the relevant sections, we will compare two different face detection algorithms. One being the Haar cascade the other one being a deep neural net. Our image comparison will be based on the predefined methods of histogram comparison. In our final result we will combine image comparison with face detection to reduce the amount of faces as much as possible without deleting persons.

Chapter 3

Proposed Solution

In the relevant work we discussed multiple methods on multiple topics. Given the material reviewed in Chapter 2, we have seen that no ready-to-use method exist for our specific task. Hence, we propose here a pipeline that aims to solve this problem. We next describe the different steps of the pipeline, as follows.
Section 3.1 describes the methods for detecting faces.
Section 3.2 explains the process of computing and comparing the histograms.
Section 3.3 puts the more detailed descriptions together and shows an overview.

The general pipeline is seen in Figure 3.1. In this image we also the stages without relevant sections. These stages are mentioned in the overview. In this figure we also note the data types that we use.

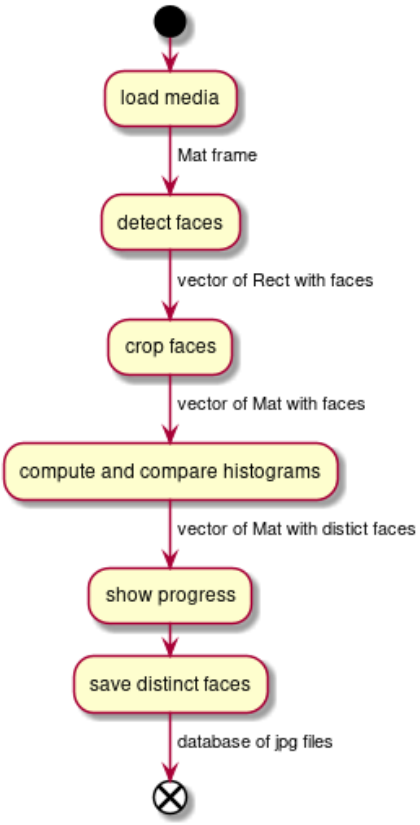


Figure 3.1: General pipeline of the proposed solution

3.1 Detect Faces

As stated in the conclusion of the relevant work, we use two different face detection algorithms and test their performance. After evaluation of the obtained results we will pick the best one to use in out final product. In the software we add an option to select the desired method. We assume in this step that we have successfully loaded a video source already. The flowchart for the whole face detection is shown in Figure 3.2.

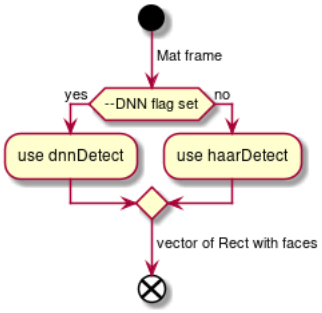


Figure 3.2: Detect face more elaborate

More detail about how dnnDetect and haarDetect work, is given in the upcoming subsections.

3.1.1 Haar Cascade

For the Haar cascade [21], we use multiple trained cascade files provided with the opencv library [2]. Not all these files are meant for face detection, we do only compare the files that are trained for face detection. The provided xml files for face detection are:

- haarcascade_frontalface_alt2.xml
- haarcascade_frontalface_alt_tree.xml
- haarcascade_frontalface_alt.xml
- haarcascade_frontalface_default.xml

In our results we remove the haarcascade_frontalface_ part for a less wide output table. If we do not choose a file as trained cascade the default file is used, being alt2.xml. We load one of these files and then call the function:

```
void cv::CascadeClassifier::detectMultiScale(
  InputArray  image,
  std::vector<Rect> &objects,
  double  scaleFactor = 1.1,
  int  minNeighbors = 3,
  int  flags = 0,
  Size  minSize = Size(),
  Size  maxSize = Size()
)
```

Here we see that there are 5 parameters with default values. For our experiments we choose to keep all the parameters at their default values. Before feeding the frame to this function we first convert the frame to gray-scale.

3.1.2 Deep Neural Net

For the deep neural net [10] we use one trained model and one specification file. The trained model has the file extension ".caffemodel". Here we have no option to set any one of these files, and thus we always use the default ones. These files are: "deploy.prototxt.txt" and "res10_300x300_ssd_iter_140000.caffemodel". An option that is able to change is the confidence level. Since we use a neural net we get a list of faces (rectangles) and there confidence. We only accept a face if the confidence is higher than the specified confidence. The function that we use to get the output layers is:

```
Mat cv::dnn::blobFromImage (InputArray image,
    double  scalefactor = 1.0,
    const Size &  size = Size(),
    const Scalar &  mean = Scalar(),
    bool  swapRB = true,
    bool  crop = true
)
```

Here we keep the scalefactor at the default. However we set the Size to the trained size being (300, 300). Also we set the mean to (104, 117, 123). And both swapRB and crop are set to false. This is done because it gave us the best results and the source for our trained model suggested it.

3.2 Compute And Compare Histograms

We use histogram comparison because it works on all kinds of images that we have and our main purpose here is to reduce the amount of repeated images. Computing and comparing the histograms of the facial images has multiple steps. First for all images that are found in a frame, we compare them with all faces in the detectedfaces vector. This vector contains all the distinct faces so far. Alse this vector has variable size and different values for this size will be tested. What this size means, the amount of faces in memory where we compare with. So lets say we use a size of 20 then each new face will be compared with the last 20 distinct faces. If we find a new distinct face we will save the last modified face and update it to the new face. With last modified we mean the longest not updated face. We update a face when a face is found the same as one in the vector. The comparison of images is done with histograms.

To compute the histogram of an image we apply the following steps:

1. Convert the colors to hsv
2. Specify the number of bins
3. Specify the color ranges

4. Compute the histogram
5. Normalize the histogram

Once this is done we also save these histograms for later comparison.

To compare two histograms (H_1 and H_2), first we have to choose a metric ($d(H_1, H_2)$) to express how well both histograms match. OpenCV [2] implements the function compareHist to perform a comparison. It also offers 4 different metrics to compute the matching. For all these metrics we have to set some settings being:

- The bins for hue and saturation, we pick 50 and 60
- The ranges for hue and saturation, we pick $[0 - 180)$ and $[0 - 256)$
- The channels, we pick $[0, 1]$

We will list in the upcoming subsections the different histogram comparisons that we will compare.

3.2.1 Correlation

The first method as metric is the correlation. The distance is given by the following formula:

$$d(H_1, H_2) = \frac{\sum_I (H_1(I) - \overline{H}_1)(H_2(I) - \overline{H}_2)}{\sqrt{\sum_I (H_1(I) - \overline{H}_1)^2 \sum_I (H_2(I) - \overline{H}_2)^2}}$$

where

$$\overline{H}_k = \frac{1}{N} \sum_J H_k(J)$$

and N is the total number of histogram bins.

3.2.2 Chi-Square

The second metric is the Chi-Square it is given by the following formula:

$$d(H_1, H_2) = \sum_I \frac{(H_1(I) - H_2(I))^2}{H_1(I)}$$

3.2.3 Intersection

The formula below defines the third metric:

$$d(H_1, H_2) = \sum_I min(H_1(I), H_2(I))$$

3.2.4 Bhattacharyya distance

As last the Bhattacharyya distance metric formulate is presented below:

$$d(H_1, H_2) = \sqrt{1 - \frac{1}{\sqrt{\overline{H}_1 \overline{H}_2 N^2}} \sum \sqrt{H_1(I) \cdot H_2(I)}}$$

3.3 Overview

Here we present a more detailed overview of the pipeline of our solution. We start with the uncovered small steps from the large pipeline, being:

- Open media, the process of opening the media stream. This is done with the `--media` flag. If this flag is not set we open the default media stream, this most of the time is your webcam.
- Crop faces, once we got a frame and a vector of rectangle from the face detection method. We make a new image corresponding to the returned rectangle. Then we resize this rectangle to a square with variable size. This size is not changeable and for the rest of the project we use a size of 128.
- Show progress, the video with the detected faces and the distinct vector is shown to the user. The output example will be shown in the results section.
- Save distinct faces, here we can also have to option to save all faces and all frames. This is done for obtaining the results that we will present in the results section. If this option is set all the frames will be saved to: `database/frames/` and all the faces will be saved to: `database/faces/`. However if the option is not set these folders remain empty. The final distinct faces are saved in: `database/out`.

Below we put all the smaller section together to get an image of the proposed solution as a whole.

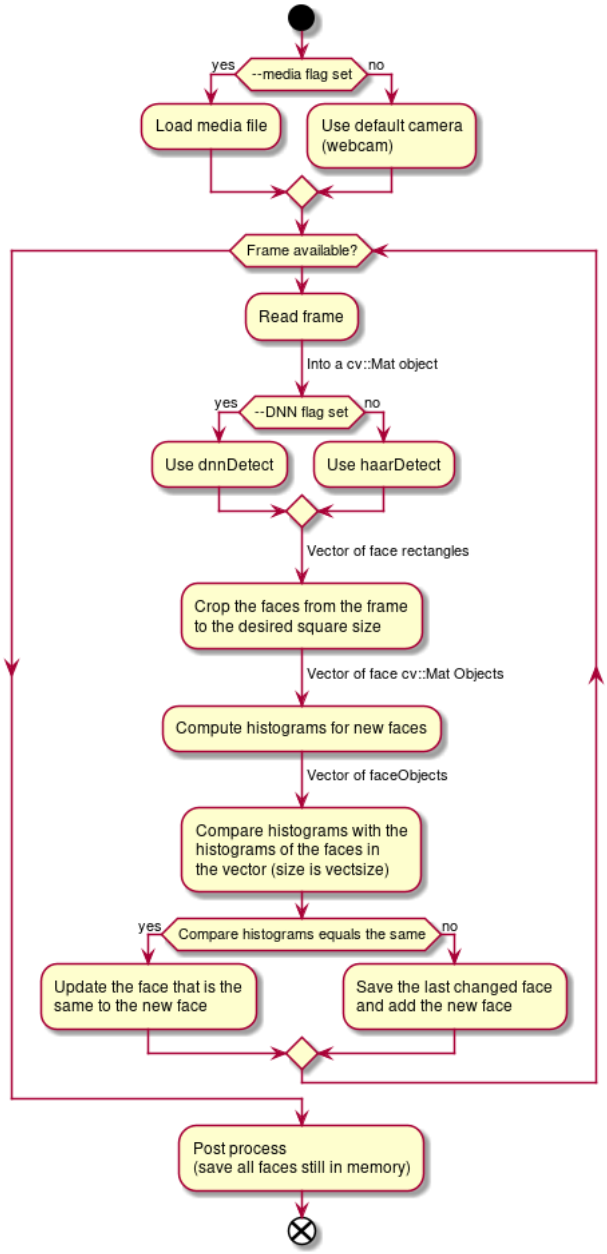


Figure 3.3: A more detailed overview of the proposed solution

Chapter 4

Results

In this chapter we present the results of applying the proposed algorithms onto multiple different videos. We have three sections with different type of results, all explaining one broad topic. First in Section 4.1 we compare multiple algorithms on detecting faces. Also we apply different configurations and parameter values. Here-after we show the results of different type of histogram comparison methods, applied to the faces detected. This is done is Section 4.2. Then we present the output of the final product in Section 4.3. We also present the software output interface. In this output both face detection and image comparison are combined to find as accurate as possible the amount of people present in a video. Our main goal for face detection is to first find as many faces as possible, with low amount of false positives. We this try to reduce the faces in the database as close as possible to the amount of persons present in the video, by the histogram techniques.

4.1 Face Detection

In this section we will present the obtained results for different types of face detection. We start this section with a table that shows an comparison of the different face detection methods. This is done in Subsection 4.1.1. In Section 4.1.2 we show one interesting frame for every video in the table. We also briefly explain the scene in the different videos.

4.1.1 Table Overview

Below we present Table 4.1 with the results for the face detection algorithms and configurations, on four videos. Of these four videos the first three marked as is4, is11 and is5 are isis videos. The last video marked as rand1 is a random video of a boxing match. All videos take approximately 4 minutes.

Method	Datafile	Number of raw faces detected				False positive (in the first 1000 images)			
		is4	is11	is5	rand1	is 4	is11	is5	rand1
Haar cascade	alt	23859	9751	7091	21318	13	77	87	103
	alt2	25923	11525	8484	23775	16	113	178	202
	alt_tree	2781	1845	32	1785	0	0	0	55
	default	25216	12635	10778	25413	248	397	230	397
Deep Neural Net	res10 conf=0.7	30950	7347	7323	3220	0	0	0	20
	res10 conf=0.4	34632	9805	8790	4823	0	1	10	136
	res10 conf=0.2	37446	12020	10598	8563	55	59	94	301

Table 4.1: Comparison of different face detection algorithms in four videos

In the Table 4.1 we see that for isis video 4 the deep neural nets performs well, since the number of faces detected is quite high while the false positives are relatively low. However if we compare this result with the result of random video 1 we see that the deep neural nets performs worse. Especially the number of faces detected is low if we compare this with the Haar cascades. Another observation is the low amount of faces detected with the `alt_tree` cascade. For example in isis video 5 this cascade only detects 32 faces, while alt2 detects 8484. This suggest that this method is quite useless, since it misses identities in the output.

We choose the evaluate the false positives in the first 1000 images since it is quite labor intensive to go trough all detected faces. What we classify as a false positive is any saved face that is not clearly seen or usable. There are some examples of corner cases here but we tried to evaluate as consistent as possible. As an example if we look at Figure 4.1, we vaguely see a face, but it is not usable and hence it is marked as false positive. Also if an zoomed in version of the face with, for example only the eyes, then this is also a false positive. Same is true for faces that are to far away in the image. Next up we will present some individual frames where the detected face are marked with green rectangles.

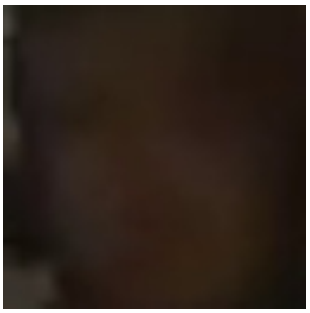


Figure 4.1: Face detection corner case

From each video an interesting frame is selected and we provide some information on what happens in that particular video. Keep in mind that from one frame it is hard to say anything about the face detection methods. These individual frames merely act as an example for understanding the performance, for comparison the table should be used.

4.1.2 Individual Frames

In this subsection we show one frames for each of the four video's. For all face detection methods we show the same frame so we can compare the results.

Isis Video 4

In isis video 4 a static scene with 8 people in total is shown. Also the camera moves slowly from left to right. One of the 8 persons present in this video covers his face, this is not detected by the face detection. This means it is already not possible to get an 100% accuracy here. Furthermore the faces are clear in the video and their resolution is quite high. We now present the very first frame of the video:

(a) alt

(b) alt2

(c) alt_tree

(d) default

(e) DNN conf=0.7

(f) DNN conf=0.4

(g) DNN conf=0.2

Figure 4.2: The first frame of isis video 4 with different face detection algorithms

In this Figure 4.2 we directly see a difference between the different methods. Both DNN conf 0.4 and DNN conf 0.2 have correctly identified all the faces in this frame, in contrast to all the other methods. Here also clearly see that no Haar cascade has identified more then 3 people, while there is no DNN that find less than four people. This suggests that the DNN performs better on this frame. In Sub-figure 4.2b we see an example of a false positive face. Another observation we make is that Sub-figure 4.2c is unable the find any faces. This also corresponds to its results shown in the table. Next we look at an interesting frame form isis video 11.

Isis Video 11

In this isis video 11 we see a leader giving a speech. There are a lot of moving object behind him and there are also many people present. It starts with an zoomed out vision of the scene while later there are more close up shots taken. Most of the face detection methods start with 30-50 false positives, because of this.

(a) alt

(b) alt2

(c) alt_tree

(d) default

(e) DNN conf=0.7

(f) DNN conf=0.4

(g) DNN conf=0.2

Figure 4.3: The 4850th frame of isis video 11 with different face detection algorithms

wd

We decided to present frame number 4850 in Figure 4.3 since it shows nine detectable faces. With detectable we mean that they are not to far away or only partially present. Here we see that the Haar cascades perform better in comparison with the previous presented frame. All Haar cascade detect eight out of nine faces except Sub-figure 4.3c However the only one detecting all present faces is Sub-figure 4.3g, being the DNN with confidence equal to 0.2. Another interesting observation is the fact that none of the methods have any false positives.

Isis Video 5

In isis video 5 a short news item is presented with multiple interviews. Almost all frames have at most one person present and there is a lot of swapping between scenes. In this video there are 5 people present, however they are spread over multiple scenes and in one scene they are presented in black and white. This means to count exactly five persons is going to be a difficult task. From this video we present the 2563th frame. We tried to find a frame where it is more clear that a low confidence level also has some disadvantages. This is since from previous examples it seems like the DNN with confidence 0.2 is the best at detecting faces. Keep in mind that overall in this video all Haar cascades have more false positives then the DNN.

(a) alt

(b) alt2

(c) alt_tree

(d) default

(e) DNN conf=0.7

(f) DNN conf=0.4

(g) DNN conf=0.2

Figure 4.4: The 2563th frame of isis video 11 with different face detection algorithms

In this figure it is clearly seen that a lower confidence also makes more mistakes. If we look at Sub-figure 4.4g we see two false positives in one frame. This only happens with a confidence level of 0.2 since in Sub-figure 4.4f we see only one false positive. Now if we would have taken a confidence level of 0.7 we would have correctly identified the face in the given frame as is seen in Sub-figure 4.4e. The Haar cascades on the other hand have no problem detecting the single person in this frame, with Sub-figure 4.4c as an exception.

Random Video 1

This video shows a summary of a boxing match. Here a lot of people are present of whom mostly in the background, or vague. The resolution of the faces present is significantly lower then those in previous videos. Counting the persons present is a difficult task since whole overviews of the stadium are shown. We present frame number 1718.

(a) alt

(b) alt2

(c) alt_tree

(d) default

(e) DNN conf=0.7

(f) DNN conf=0.4

(g) DNN conf=0.2

Figure 4.5: The 1718th frame of rand video 1 with different face detection algorithms

In this Figure 4.5, we see clearly that the Haar cascades preform better then the DNN methods. The DNN methods are not able to detect any face in the given frame, while the Haar cascade method alt detects nine faces, as seen in Sub-figure 4.5a. This difference is noticed from the early presented table where the Haar cascades in total detect more faces then the DNN methods. So this means that when a low resolution crowd in the background is present the Haar cascade is actually preferred.

4.2 Image Comparison

In this section we will present the results of comparing the four histograms described in the proposed solution. We start by giving a table overview 4.2.1. Here all methods are compared on 20 sets of three face images. The next Subsection 4.2.2 shows some of the 20 images presented in the table. As last we present a plot for each of the compared histogram methods, in Subsection 4.2.3.

4.2.1 Table Overview

Here we present the Table 4.2 with all the outputs of the different histogram comparison methods. In this table we compare two pair of pictures. One pair are two faces that are the same and one pair are two different face images. With the same image we mean neighboring frames of the same person. Al pairs of three images are always from the same video, this improves the relevance of our results to the project. This can be more clearly seen from the examples given in Subsection 4.2.2. We also made all the extreme values bold face to make it more clear for the reader. In this table the first 10 images are of resolution 256*256 pixels, the last 10 images are 128*128 pixels in resolution. The images are obtained by the detected faces phase. The first 3 image sets are from isis-videos as well as the images sets numbered from 10 till 16. The rest of the images are obtained from the random videos.

Method	Correlation		Chi-Square		Intersection		Bhattacharyya distance	
Image	same	different	same	different	same	different	same	different
1	0.963060	0.608164	23.264543	88.050640	20.500527	9.792500	0.235165	0.576172
2	0.988773	0.656220	2.666243	151.016672	15.743347	8.743163	0.136703	0.635783
3	0.986752	0.744598	5.814792	156.621258	20.542990	13.284405	0.152532	0.502573
4	0.998583	**0.223442**	0.304163	310.269375	9.477780	3.734679	0.075555	0.625890
5	0.995940	0.635320	0.937772	21.426114	22.124220	7.613316	0.068103	0.513184
6	0.989958	0.274751	1.216002	270.266692	16.964873	7.092532	0.105071	0.644647
7	0.980178	0.655459	5.841359	51.878833	22.170627	12.827869	0.139945	0.501622
8	0.988142	0.477425	1.348458	157.466081	11.110056	7.540237	0.087968	0.546642
9	0.981464	**0.911165**	1.872143	16.319383	12.144464	6.850345	0.101900	0.357090
10	0.981981	0.645688	1.140025	32.009531	13.766828	6.772082	0.111970	0.440944
11	0.984352	0.322726	2.872238	131.460977	12.577388	4.768708	0.169069	0.579799
12	0.949547	0.379059	**25.334920**	31.741039	28.088284	5.736572	**0.247033**	**0.633223**
13	0.966749	0.470153	8.660361	**389.798948**	**30.815043**	**18.957516**	0.164106	0.591578
14	**0.947601**	0.538319	7.277235	104.365186	16.354882	12.642059	0.209112	0.566071
15	**0.998818**	0.606611	**0.150347**	22.835556	23.928133	9.932561	**0.039966**	0.467661
16	0.994577	0.613956	0.638901	44.499219	**8.198251**	6.353394	0.103678	0.447084
17	0.994926	0.677362	0.796749	11.774120	14.011228	7.827896	0.090698	0.375195
18	0.997855	0.549256	0.672146	38.942771	11.189976	6.716741	0.114651	0.410974
19	0.998483	0.275613	0.467269	50.544479	8.409184	**3.520963**	0.117601	0.579656
20	0.995277	0.859771	1.413799	**9.817797**	18.985585	10.810241	0.126748	**0.306647**

Table 4.2: Histogram Comparison techniques

Our first observation is that in two of the four methods, we can draw a boundary between the different and the same images. This is seen in the Correlation (low=0.947601, high=0.911165) and in the Bhattacharyya distance (low=0.306647, high=0.247033). In the other two methods there is clear overlap, being Chi-Square (low=9.817797, high=25.334920) and Intersection (low=8.198251, high=18.957516). This means if we would pick one of the overlapping methods, we would then lose faces that are not the same, or end up with more faces then desired. This means that the to the none overlapping histogram methods probably with preform better.

In the Correlation and the Intersection methods, a high value means that two images are the same, while in the Bhattacharyya distance and in the Chi-Square methods a higher value means that two images are different. Some results that are worth mentioning are image 12 since it got 3 bold face values, same is true for images 13 and 15. Another interesting fact is that in the first 10 images only two extreme values are found. This could suggest that a higher resolution has impact on the performance of the histogram methods. We do however use a resolution of 128*123 for the rest of the results.
Next we present the images with some of the interesting results.

4.2.2 Image Examples

In this subsection we show four pairs of three images, that gave interesting results. We start with the images 12, 13 and 15 since, as stated above, they all have 3 extreme values. We talk in this Subsection about images be "least/most the same/different". If we for example say these images are marked most the same by the Chi-Square method, we mean that the base and the same images are marked with the extreme value of being most the same. As stated before if this is a high or low value depends on the method. What is most interesting is the least the same and the least different since those cases have influence on the later given threshold value.

Figure 4.6: Base

Figure 4.7: Same

Figure 4.8: Different

Figure 4.9: Comparison of image 12

The results on these three images seen in Figure 4.9 are, that they are least the same using the Chi-Square and the Bhattacharyya distance. Also they are most different in the Bhattacharyya distance. Especially the fact that the base and same images are least the same is remarkable. It looks like the images are only shifted a small distance vertically.

Figure 4.10: Base

Figure 4.11: Same

Figure 4.12: Different

Figure 4.13: Comparison of image 13

In Figure 4.13 we see the images corresponding the image 13 in the Table 4.2. The Chi-Square classifies this image set as the most different. The Intersection classifies this image set as least the same, and least different. These are quite interesting since if we look at the differences between the base and the different image, we can clearly see the difference. For example the person in the different image wears sunglasses while the person in the base image does not. And this time the difference between the base and the same images is only a small shift horizontally. Now we look at image 15:

Figure 4.14: Base

Figure 4.15: Same

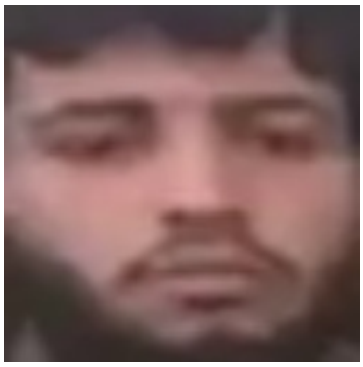


Figure 4.16: Different

Figure 4.17: Comparison of image 15

This image set seen in Figure 4.17 is rated as most the same in three of the four methods. It is excepted since there is almost no movement of the face. The different image does not result in any extreme results. Now we look at image 20 since it is marked least different in two of the four methods. This means we expect that the base and different images look a lot like each other.

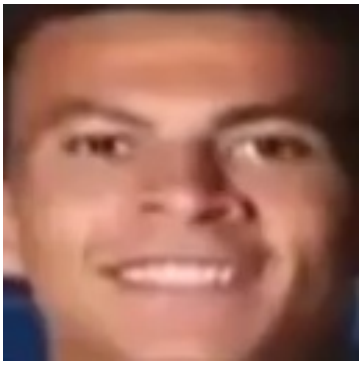


Figure 4.18: Base

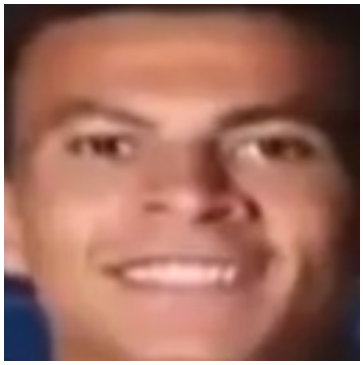


Figure 4.19: Same

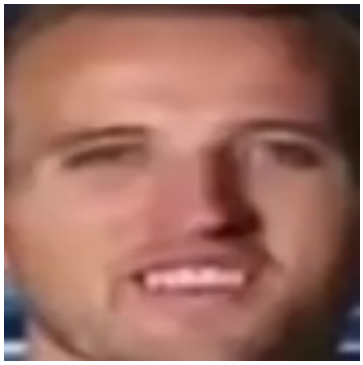


Figure 4.20: Different

Figure 4.21: Comparison of image 20

It is indeed seen in Figure 4.21 that the base and different image look a lot like each other. Almost the whole picture is filled with the face so there is no difference in background. Furthermore the skin color is also quite similar so for our histogram methods it looks quite the same. As last we look at an image with no extreme values, a "normal" case.

Figure 4.22: Base

Figure 4.23: Same

Figure 4.24: Different

Figure 4.25: Comparison of image 6

And indeed in Figure 4.25 we see a different person in a different position. Hence explaining that we find no extreme values here.

4.2.3 Histogram Plots

In this Subsection we present four plots for each of the four histogram methods. In these plots the blue dots, are the base-same scores, we will refer to these points are the same images/line. The gold crosses are the base-different scores, we will refer to these points as the different images/line. There is also a stripped line, the boundary line. This line is calculated as follows:

$$\frac{\texttt{least same + least different}}{2}$$

The exact value of this boundary value is given in the title of the corresponding plots. The purpose of showing these plots is to make it more clear on the different performance for the image comparison methods

Correlation

Here we show the plot of the correlation method.

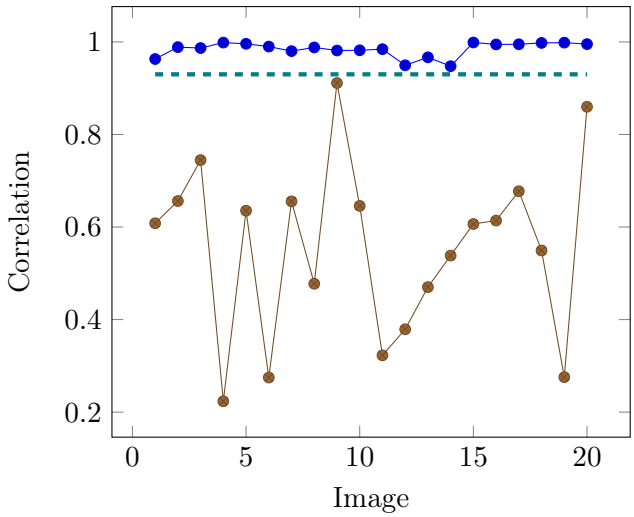


Figure 4.26: Correlation, boundary at 0.93

In Figure 4.26 we see that the boundary separates the same images from the different ones. Another observation from the figure is that there is a lot of fluctuation in the different line, while the same line is almost constant. This is expected, since the same images are always neighboring frames while the different images are just random so more fluctuation in the different images is to be expected.

Chi-Square

Below we show the plot of the Chi-Square distance measure.

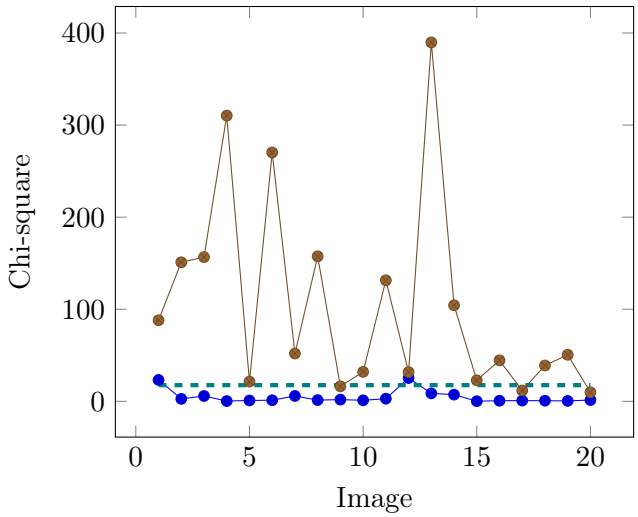


Figure 4.27: Chi-Square, boundary at 17.57

Here in Figure 4.27, we see an example of the boundary line intersecting the same and different lines. Again a lot of spread in the different line while the same line seems more stable.

Intersection

Next we present the plot for the Intersection method.

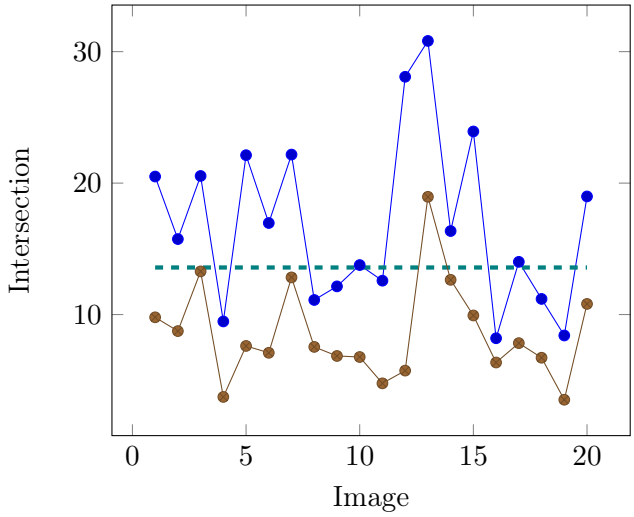


Figure 4.28: Intersection, boundary at 13.58

For this measurement the points in Figure 4.28 seem to be spread. This is true for both the same line as well as for the different line. It is hard to see where the boundary line should be because of this. This also shows that this method probably is not very effective for comparing facial images.

Bhattacharyya Bistance

For the last plot, the Bhattacharyya distance is presented

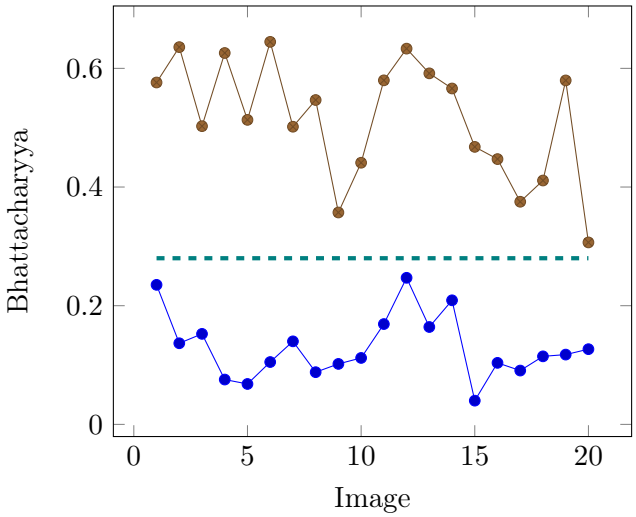


Figure 4.29: Bhattacharyya, boundary at 0.28

In this Figure 4.29 we see a very clear boundary. The points are spread for both the difference and for the same lines. However because of this clear boundary this method is most favorite and will probably work the best.

4.3 People Count

In this Section we are going to present the final results of our project. First we compare different vector sizes and histogram methods. This is done in Subsection 4.3.1. Once we have shown this table we continue with a comparison between the face detection methods, in Subsection 4.3.2. As last in Subsection 4.3.3 we present a table showing what different threshold values do with the output. In these results we pick one video with many people present (crowded) and also one video that is more static with a countable number of persons present. After we have seen the best vectsize, histogram method, face detection method and threshold we pick the best values and apply this to all isis videos. The results for these videos is shown in Subsection 4.3.4. In all these tables we present the output of our program, meaning all the distinct faces present in the video. To give some understanding of the software output we also present one example in Subsection 4.3.5 In all tables we accept a face to be the same if the images are equal or more the same than the boundary.

4.3.1 Different Histogram Methods

This Subsection shows a comparison of the different histograms methods and different vector sizes. We present Table 4.3, where we have picked two videos for comparison. In isis video 4 a static scene is shown and in isis video 6 many people are present. For all the threshold values we pick the boundaries shown in the plots for each method. This corresponds to 0.93 for Correlation, 17.57 for Chi-Square, 13.58 for Intersection and 0.28 for the Bhattacharyya distance. The vector size corresponds to how many faces we remember to compare with. So if the vector size is 20 (the default value) we compare with the last 20 faces.

Video	isis 4 (34632 faces)			isis 6 (66371 faces)		
Histogram method	Vector size 10	20	40	Vector size 10	20	40
Correlation	1893 (5.46%)	1873 (5.41%)	1850 (5.34%)	9985 (15.04%)	2624 (3.95%)	2516 (3.94%)
Chi-Square	1960 (5.66%)	1802 (5.20%)	1695 (4.89%)	9689 (14.60%)	3651 (5.50%)	3270 (4.93%)
Intersection	3423 (9.88%)	3278 (9.46%)	3181 (9.18%)	10808 (16.28%)	10023 (15.10%)	9756 (14.70%)
Bhattacharyya	1050 (3.03%)	1045 (3.01%)	1046 (3.01%)	10777 (16.24%)	2009 (3.03%)	1927 (2.90%)

Table 4.3: The amount of distinct faces saved for several histogram methods and vector sizes. We use the DNN with confidence 0.4 for detection. All of the tested algorithms had at least one face of each person in the video in the output.

What is clearly seen is that in the video where only 8 persons are present (isis 4), the vector size does not have a significant influence on the output. On the other hand if we look at the more crowded video we see a clear drop in detected persons. For the Bhattacharyya it drops from 10777 distinct face to 2009 distinct faces, this is an 81% drop. However the trade-off in performance for making the vector size 40 does not seem worth it, since it only improves the output a little. We also observe that the Bhattacharyya distance method has general speaking the best performance. This is what we expected when we analyzed the graphs for each histogram comparison method. For both videos this method reduces the database with roughly 97%. As last we see that the Intersection method shows the worse results, this was also expected if we look at the graph of this method.

4.3.2 Multiple Detection Methods

This Subsection shows the different face detection algorithms, configurations on the final output. We show Table 4.4 were the distinct persons are presented. We also added percentages and averages to make the results more clear. The cascade `alt_tree` is not covered since it had a very low detection rate. In this graph the crowded video is isis video 11 and the more static video is isis video 7.

Face Detection	DNN 0.7		DNN 0.4		DNN 0.2		Alt		Alt2		Default	
Video	Total	Distinct	Total	Distinct	Total	Distinct	Total	Distinct	Total	Distinct	Total	Distinct
isis 7	10668	165 (1,55%)	11727	237 (2.02%)	15012	447 (2.98%)	8511	348 (4.09%)	7542	288 (3.82%)	13309	818 (6.15%)
isis 11	7347	261 (3.55%)	9805	375 (3.82%)	12020	724 (6.02%)	9751	992 (10.17%)	11525	1290 (11.19%)	12635	1861 (14.73%)
Average	9007.5	213 (2.36%)	10766	306 (2.84%)	13516	585.5 (4.33%)	9131	635 (6.95%)	9533.5	789 (8.28%)	12972	1339.5 (10.33%)

Table 4.4: The amount of distinct faces saved for different face detection methods. We use the Bhattacharyya distance as comparison method, with confidence 0.28 and vectsize is equal to 20.

From this table we can again clearly see difference in performance between the deep neural nets and the Haar cascades. All deep neural nets have at a reduction of less then 5% with the DNN with confidence 0.7 performing the best having a reduction of 2.36% on average. While the Haar cascades on the other hand all score higher then 6% on average. We also observe that all methods perform better on isis video 7 then on isis video 11. While the DNN with confidence 0.7 performs the best, we still choose the DNN with confidence 0.4. This is because if only performs a bit less (2.84% instead of 2.36%) while it got significant higher detection rates.

4.3.3 Variant Threshold Levels

The last setting we want to test before presenting the final output, is the influence of threshold values. We test three different threshold values on the Bhattacharyya distance histogram method. We pick three values, the least the same value (0.25), the least different value (0.30). These both come from the histogram comparison table. As last value we pick the actual earlier seen boundary value (0.28). Again in this table we have a crowded video (isis 9) and a more static video (isis 5).

Video	Threshold		
	0.25	0.28	0.3
isis 5 (7323 faces)	150	93	74
isis 9 (30018 faces)	803	552	444

Table 4.5: The distinct people for three threshold values from the Bhattacharyya distance. For detection we use the DNN with confidence 0.7.

From the Table 4.5 we see that, as expected, a higher threshold ends up with less distinct faces. However the difference between 0.25 and 0.28 is quite high in comparison with the difference between 0.28 and 0.3. Choosing a higher threshold increases the risk of losing

faces. Because of this risk and not very significant improvements in output, we choose to go for a threshold of 0.28.

4.3.4 Final Results

Here we present the earlier discussed "best" settings, on all the given videos. Before testing the final settings, we check if for all the videos that have a countable number of persons in them, at least one face images is still present in the output. Lets take video 4 for example, here 7 different persons are detected, so at least one picture of all these 7 person should be present in the output folder. We did this check for our final settings being, vectSize = 20, DNN conf=0.4, Bhattacharyya threshold=0.28, on all videos with countable amount of persons. These videos being isis video 4,5 and 7. Our method passed all these test and thus does not delete any identity.

Video	Total faces	Distinct faces
Isis 4	34632	1045 (2.93%)
Isis 5	8790	148 (1.68%)
Isis 6	66371	2009 (3.03%)
Isis 7	11727	237 (2.02%)
Isis 8	31841	969 (3.04%)
Isis 9	36757	840 (2.29%)
Isis 10	19970	1738 (8.70%)
Isis 11	9805	375 (3.82%)
Isis 12	9593	802 (8.36%)
Rand 1	4823	558 (11.57%)
Rand 2	1332	96 (7.21%)
Rand 3	28647	3726 (13.01%)
Rand 13	44438	1613 (3.63%)
Rand 14	2201	258 (11.72%)
Average	22209.07	1029.57 (4.64%)

Table 4.6: The number of distinct people counted on all the videos in the database. As settings a vectSize of 20 is used, for detection we use the DNN with confidence 0.4 and for comparison the Bhattacharyya distance with threshold equal to 0.28 is used.

From this table we see that we obtain an average reduction of 95.36%. We have an average detection of 22209.07 faces per video. This result could be even lower with a higher threshold for the Bhattacharyya distance, or if we use a higher vector size. However the trade of in performance is not worth it as earlier discussed. Again we see that the solution works better on the isis video in comparison to the random videos. This can be explained by the fact that all settings are trained on the isis videos since they are mote relevant to the problem we try to solve.

4.3.5 Program Output

Below the present an image of the real time output of the program.

Figure 4.30: An example of the program real time output.

Here we see in the upper half of the image the running video with the faces detected. In the bottom half we see the different faces detected. The vectsize is clearly 20 in this images since we see 20 different images. The images that are being updated seem to move in the bottom half. Another observation we make is that some images in the distinct vector look quite similar. For example the person with the red hat has three distinct faces that all look like each-other. This phenomenon is also seen in the examples we gave in the histogram comparison methods. Here we saw that a small shift vertically or horizontally can cause the images to look different in therms of the histogram method.

Chapter 5

Conclusion

In this thesis, we have researched and implemented different methods for face detection, image comparison and combined them to count people in videos. The aim was to detect all persons in a video of low resolution and to reduce the facial images that we obtain without deleting any identity. In order to test our solutions and obtain the results, we made an command line software program that clearly shows it progress and is highly configurable. Next we review of the original goals and requirements, and in what proximity we achieved them.

Research question 1: How many different people are present in the video collection?
After a literature study, we found the two different methods for detecting faces. From these methods we choose to compare two of them being the Haar cascade and the deep neural net. We discussed multiple techniques to compare these faces and choose to use histograms. After evaluating the results we conclude that a Deep Neural Net is preferred over the Haar cascade and that the Bhattacharyya distance is the best method for comparing facial images. With these methods we reduce the initial obtained facial images with 95.36% on average.

Research question 2: How to group faces (present in the individual frames) per person, so that we can tell where and in which specific frames the same person appears?
In our literature study we found multiple solutions to this problem, however we did not implement or test any in our program or thesis. This has to do with the complexity of these methods in combination on the low amount of results on low resolution images. This could be the next step for future work, to group the output we obtained and try to come even closer to the amount of persons present, without losing any identities.

Research question 3: How to select a good "representative face" for the above groups? A representative face is one of good pose (orientation), resolution, lighting, contrast, and image sharpness.
This question is not answered in this thesis. However it would be nice to use the results obtained from this project to help solve this question, it can be an project on its own if we take into account the estimated time it will take. We can select images with higher resolution and detect whether an image has blur or not, still that is not enough for automatic selection of the most representative face.

The algorithms can be demonstrated and used in a program we built. The requirements for this software were: video reading, selection and adjustment of algorithms and parameters to use and show the progress in a organized fashion. All these functionalities were successfully implemented. A practical requirement was that video could be previewed and that the distinct faces could be shown in real-time. The upcoming section is about the limitations 5.1 then we talk about future work 5.2.

5.1 Limitations

In the upcoming list we presents aspects where research was limited or could be improved.

- Time performance of applied methods. We did not measure nor talk about the time that the program took for different face detection algorithms. This could change the conclusion about which of the methods is most applicable.
- The amount of tested videos. Our problem states that we try to automate operations on large databases of video sources, however we only tested out results on sixteen videos. This can also mean that the preferred settings on these videos do not apply in general.
- Proper comparison with low resolution face recognition. We did not have the time to implement or test face recognition, this could help to verify that indeed it does not properly work for our problem.

5.2 Stimulus For Future Work

From the earlier conclusions it follows that we only partially answered research questions one. Here is the first stimulus for future work, to improve om my work to come closer to the actual amount of persons present in the presented videos. The other two research questions are also future work challenges. To cluster the received facial output and to recognize the persons present would improve the usability of our research.

On the scientific aspect, it should be noted that during this project we have worked with several existing methods from the related literature, on the topic of face detection. We made comparison between two available techniques and evaluated the results. However for the image comparison we only used one technique being the histogram method, here improvements can be made to try other kind of image comparison techniques.

On the software aspect, many improvement can be made. For example an actual user interface can be build, so that it is possible for non command line users to understand the program. Furthermore implementing more image comparison techniques or adding post-processing can improve the output.

Bibliography

[1] Face detection. https://en.wikipedia.org/wiki/Face_detection. Accessed: 2018-06-24.

[2] Opencv library. http://opencv.org. Accessed: 2018-05-09.

[3] Welcome to the surveillance state: China's ai cameras see all. https://www.huffingtonpost.com/entry/china-surveillance-camera-big-brother_us_5a2ff4dfe4b01598ac484acc?guccounter=1. Accessed: 2018-05-31.

[4] Ziuz holding b.v. http://www.ziuz.com/nl. Accessed: 2018-06-27.

[5] F. Brokken. icmake v9.02.07, copyright (c) gpl. https://github.com/fbb-git/icmake, 1992–2018.

[6] F. Brokken. bobcat v4.08.03, copyright (c) gpl. https://github.com/fbb-git/bobcat, 2005–2018.

[7] V. Di Gesu and V. Starovoitov. Distance-based functions for image comparison. *Pattern Recognition Letters*, 20(2):207–214, 1999.

[8] C.-h. DU, Z. Hong, L.-m. LUO, L. Jie, and X.-y. HUANG. Face detection in video based on adaboost algorithm and skin model. *The Journal of China Universities of Posts and Telecommunications*, 20:6–24, 2013.

[9] B. M. Evan Spiegel and R. Brown. Snapchat, proprietary software. www.snapchat.com, 2011–2018.

[10] S. S. Farfade, M. J. Saberian, and L.-J. Li. Multi-view face detection using deep convolutional neural networks. In *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, pages 643–650. ACM, 2015.

[11] M. A. Hmani and D. Petrovska-Delacrétaz. State-of-the-art face recognition performance using publicly available software and datasets, March 2018.

[12] W. Jia, H. Zhang, X. He, and Q. Wu. A comparison on histogram based image matching methods. In *Video and Signal Based Surveillance, 2006. AVSS'06. IEEE International Conference on*, pages 97–97. IEEE, 2006.

[13] M. F. Karaaba. *Face recognition in low-resolution images under small sample conditions with face-part detection and alignment.* University of Groningen, 2016.

[14] L. Lei, D.-H. Kim, W.-J. Park, and S.-J. Ko. Face recognition using lbp eigenfaces. *IEICE transactions on Information and Systems*, 97(7):1930–1932, 2014.

[15] C. I. Orozco, F. Iglesias, M. E. Buemi, and J. J. Berlles. Real-time gender recognition from face images using deep convolutional neural network, Nov 2017.

[16] C. Otto, B. Klare, and A. K. Jain. An efficient approach for clustering face images. In *Biometrics (ICB), 2015 International Conference on*, pages 243–250. IEEE, 2015.

[17] C. Otto, D. Wang, and A. K. Jain. Clustering millions of faces by identity. *IEEE transactions on pattern analysis and machine intelligence*, 40(2):289–303, 2018.

[18] M. Paul, S. M. Haque, and S. Chakraborty. Human detection in surveillance videos and its applications-a review. *EURASIP Journal on Advances in Signal Processing*, 2013(1):176, 2013.

[19] F. Schroff, D. Kalenichenko, and J. Philbin. *Facenet: A unified embedding for face recognition and clustering.* IEEE, 2015.

[20] M. R. Stevens and J. R. Beveridge. Image comparison techniques in the context of scene refinement. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 1, pages 685–688. IEEE, 2000.

[21] P. Viola and M. J. Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

Appendices

Appendix A

Software Documentation

The software developed for this project is intended to be usable for everyone interested to use and to improve. We provide some documentation to at least be able to use the designed software. The program is made on a linux environment due to its dependencies and philosophy it is only supported on an unix like environment. Note that we did not include detecting wrong inputs since this is more of a final step and not necessary for our purposes.

A.1 Technology stack

We choose to make the program in the C++ programming language. The OpenCV [2] library supports only two languages, C++ and python and due to the experience in C++ this was the best choice. The program for now only has command line usage of which it is described below. This can be extended to create an actual GUI, for example with QT. We use the icmake [5] tools to maintain and build the project, also we use the Arg class from the bobcat library [6] to parse the command line arguments in an structured way. This software and library are both available in all major linux distributions.

A.2 Requirements

All requirements for the developed software are listed below. We first list the output of the `people_count -h` command, we can use these explanations for the requirements.

```
people_count by Twan Schoonen
people_count V0.00.10 2018

Usage: people_count [options]
Where:
```

```
[options] - optional arguments (short options between
   parentheses):
   --help (-h)       - provide this help
   --version (-v)    - show version information and
      terminate
   --debug (-d)      - show debug information
   --media file      - examine the file for faces, if not
      specified standard input is used
   --DNN             - Use the Deep Neural Net instead of
      the Haar cascade
   --DNNconf         - specify the confidence for the DNN
      default = 0.7
   --cascade file    - if using the Haar cascade can be
      used to specify the path of the .xml file
                       the default path: res/haarcascades/
                          haarcascade_frontalface_alt.xml
   --detect          - only detect faces from input
   --saveraw (-s)    - save all the detected faces and
      frames in the database folder
   --speed (-S)      - set waiting times between frames 0
      means waiting for a click
   --clear (-r)      - remove all files in the database
      folder
   --hist            - specify the histogram method where 0
       = Correlation, 1 = Chi-Square
                       2 = Intersection and 3 =
                          Bhattacharyya distance, the
                          default is 0
   --histconf        - specify the histogram confidence
      default is 0.5
   --vectsize        - the amount of faces that is kept in
      memory to compare with default = 20
```

The user can load a media source.
This is achieved with the `--media` options. This option lets you specify the file to be read. If this is not specified the standard device is used, this most of the time is the webcam.

The user can select different face detection methods.
This is realized with the `--DNN` options. If this option is set the deep neural net is used with as confidence 0.7. The user can change this by using the `--DNNconf` option. However using this option when the Haar cascade is selected means that it get ignored. Using the Haar cascade means that we can specify an alternative cascade file with the `--cascade` option.

The user can select different histogram distance methods.
With the options `--hist` and `--histconf` we are able to select the desired histogram method and to set the threshold. Hence this requirement is satisfied.

The user can save all faces and frames with rectangles.
This functionality is in particular useful when someone only whats to use my face detection methods and work with the output of this process. It can be realized by setting the `-s` flag. When this flag is set all frames and face will be stored in the database. If the user desires to only use the detection part then the `--detect` option can be set. Another option that will help the user manage the database is the `-r` option, this cleans the database before running.

The user clearly sees what happens.
This requirement is achieved by the output shown in the results Section. We see a the video with all detected faces in the top half of the screen. In the lower half of the screen we see the progress by the presented vector of distinct faces.

A.3 Build instructions and user interaction

To build the program from source, install the dependencies get the code, go to the source folder and run:
`$ icmbuild`
This recognizes the icmconf file and builds all the classes. After the process was successful the program is located at `/tmp/bin/binary`.

An example of output when the program is run for the first time is shown below.

```
1 Using Haar cascade for detection
2 With as cascade: res/haarcascades/haarcascade_frontalface_alt
    .xml
3 Using histogram method: 0, with conf = 0.5
4 Face Detection Started...
5 Press q to quit the window.
6 [ INFO:0] Initialize OpenCL runtime...
```

In line 1 the used face detection method is printed.
In line 2 the program shows the read cascade file.
In line 3 the selected histogram method is shown with the threshold value, of course if the `--detect` flag is set then this is not printed.
The other lines are self explanatory.

Then when the program is finished we see the upcomming lines

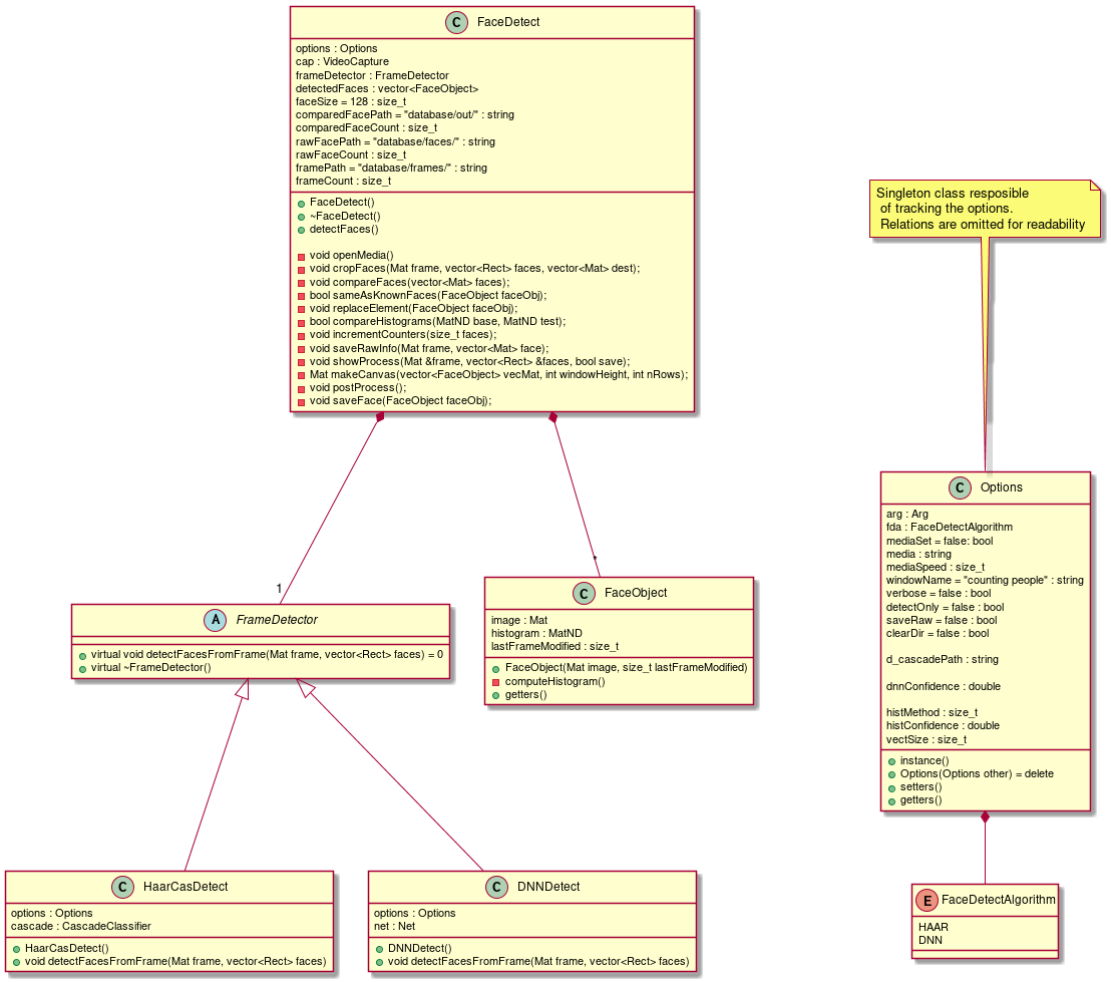
```
1 Face Detection done!
2 Found: ... faces and saved: ... distinct faces
```

Where in line 2 the `...` are numbers that actually present the performance.

A.4 Software design

In this Section we first present the total class diagram, hereafter we add some explanation. For readability we did not add any namespaces, const's or references (&).

Figure A.1: Class diagram of the software



In Figure A.1 we see the complete overview of the class diagram of the created software. The main responsibility is handled by the class `FaceDetect`. Main calls the `detectFaces()` function and then the rest of the program flow is handled by this class. What is nice to notice is that we added an abstract base class for the different face detection methods. This makes it possible to extend he program with another face detection method without too much effort. This functionality is a nice feature for the histogram comparison as well however since we did only compare multiple distance functions and not actually add multiple methods this is not implemented.