University of Zurich^UZH

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Online Identity Verification using Face Recognition

Aline Schaufelberger
Zurich, Switzerland
Student ID: 17-701-152

Supervisor: Dr. Bruno Rodrigues, Eder Scheid, Prof. Dr. Burkhard Stiller
Date of Submission: August 26, 2021

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

Zusammenfassung

Die rasante Digitalisierung der Gesellschaft führt zu erheblichen Veränderungen in zahlreichen Bereichen wie Wirtschaft, Kommunikation, zwischenmenschliche Beziehungen und auch in der Art und Weise, wie Lehr- und Lernaktivitäten durchgeführt werden. Diese Umstellung hat natürlich sowohl Vor- als auch Nachteile. Während der Unterricht in Schulen und an Universitäten relativ einfach online abgehalten werden kann, gestaltet sich die Durchführung von Prüfungen komplizierter. Bei einer gewöhnlichen Prüfung wird die Anwesenheit der eingeschriebenen Studenten anhand einer Anwesenheitsliste und des Studentenausweises überprüft, bei einer Online-Prüfung wird die Überprüfung der Studenten durch die Einschränkungen der heutigen Technologie erschwert. Daher sind Online-Prüfungen ein Bereich, in dem die Authentifizierung von Personen im Bildungssystem an Bedeutung zunimmt.

Diese Arbeit konzipiert, implementiert und testet einen Prototypen für eine Applikation zur Identifikation von Teilnehmern an Online Videoveranstaltungen wie z.B. Studenten an einer Prüfung. Diese Applikation soll dem Prüfungsorganisator die Identität der Teilnehmer offenbaren und die abwesenden Studenten ermitteln. Dazu wird eine Methode zur Gesichtserkennung verwendet, welche die Gesichter der Anwesenden Personen mit Bildern aus einer simulierten Universitätsdatenbank vergleicht.

Abstract

The rapid digitization of society is leading to significant changes in numerous areas such as business, communication, interpersonal relationships, and also in the way teaching and learning activities are conducted. This change has both, advantages and disadvantages. While classes can delivered online relatively easily, the organization of exams and assessments becomes more complicated. In a traditional exam, the presence of enrolled students is verified using an attendance list and student ID. In an online setting, student verification is aggravated by the availability and limitation of technology. Hence, online exams are a field where individual authentication becomes more significant in the educational system.

This thesis conceptualizes, implements, and tests a prototype for an application that automatically verifies the identity of individuals participating in an online video conference, such as students in an exam. This application should reveal the identity of the participants to the organizer and identify the absent students. To do this, this application uses a facial recognition method that compares the faces of the present individuals with the photos stored in a simulated university database.

Acknowledgments

I would like to take this opportunity to express my gratitude to my supervisors Dr. Bruno Rodrigues and Eder Scheid, who were always available to help me with questions or problems. Especially, I want to thank Dr. Bruno Rodrigues for his guidance during the regular meetings in the last six months.

Naturally, I would also like thank Prof. Dr. Burkhard Stiller from the Communication Systems Group (CSG) of the University of Zurich who made this thesis possible and entrusted me with this work.

In addition, I would like to express my appreciation to my family and friends, who helped and supported me during this time. It is thanks to their patience and assistance, that I have reached this point in life.

Contents

Chapter 1

Introduction

The fast-paced digitization in society (*i.e.,* digital transformation) leads to massive changes in multiple areas such as economics, communication, interpersonal relationships, and also in the way in which teaching and learning activities are conducted. Although educational institutions aim to preserve a good relationship between tutors and apprentices, a society increasingly digitized and pressured by technological advances imposes an adaptation in traditional teaching approaches with greater use of media and digital content.

The continuum of technology-based teaching ranges from pure face-to-face lessons to courses delivered entirely online. A few years ago, every teacher or instructor had to decide where on this continuum his course should be [2]. However, due to the pandemic measures, the lecturers were deprived of this decision. Within a brief period, lectures were held online with the help of video conferencing tools such as Zoom or Microsoft Teams. Naturally, this shift has both, advantages and disadvantages. On one hand, online teaching is flexible and easily accessible [17]. For many students, this has opened new ways of studying where they can manage their lectures completely independently. On the other hand, students have an increased lack of social interaction during online courses. This is a crucial factor, especially for young students. The exchange of information among students is vital and creating a network is essential for their study success and future careers. In addition, it is often the case that students will create bonds at university that can last for life.

While classes can be delivered online relatively easily, the organization of exams and assessments becomes more complicated due to the difficulty of verifying the exam setup, as well as identifying the students. In a traditional exam, the presence of enrolled students is checked by means of an attendance list and the students identification card. In an online setting, student verification is aggravated by the availability and limitation of technology. Hence, online exams are a field where individual authentication becomes more significant in the educational system.

1.1 Motivation

Verifying the identity of students in small online settings can be done within a short time with the assistance of helpers. Verifying the identities of classes with several hundreds of students would require a large number of helpers. Further, the camera quality of some student laptops would not allow a helper to read a shown identification card. This problem can be remedied by a program that automatically verifies the identities using facial recognition.

1.2 Thesis Goals

The goal of this thesis is to test and implement a prototype for an application that automatically verifies the identity of people participating in an online video conference, for example, students in an examination. It shall reveal the identity of the participants to the organizer and derive the absent students. To do so, the prototype is detecting faces on the screen and afterwards matches those faces to the reference pictures of the students provided by the university database.

1.3 Methodology

To find an optimal solution for the described problem and to build a prototype, a theoretical background needs to be established. This includes bibliographical research, in which existing methods are analyzed in detail and suitable technologies are identified and compared. The acquired knowledge is later be used for the practical part of this thesis. The following sequence of tasks allow a structured workflow:

1. **Face recognition analysis:** In order to obtain comprehensive knowledge about face recognition, the current state of the art technologies have to be investigated first. Face recognition can be divided into different sections and for each section, the available methods have to be analyzed and compared.

2. **Definition of requirements:** In this step, the requirements for the prototype are defined from a user point of view followed by a translation into design requirements. This also includes defining clear borders of what the prototype should do and what it should not do.

3. **Prototype development:** With the previously defined specifications, a prototype can be developed. However, before the prototype is implemented, the design of the application must be conceptualized.

4. **Prototype evaluation:** In the last step, the developed prototype is examined and evaluated in detail. This includes comparing the performance of the different algorithms in terms of accuracy, speed, and computational cost. To conclude this evaluation, the system is tested in a simulated examination environment.

1.4 Thesis Outline

This thesis is based on existing tools and methods that are used for face recognition. To gain a better understanding, **chapter 2** provides an overview of the different concepts and state of the art methods that are of great importance for this thesis. This is followed by the prototype design and implementation in **chapter 3** where the underlying structure and implemented code is examined in more detail. An evaluation of the methods as well as a simulated test run, is shown in **chapter 4**. In the final chapter, **chapter 5**, there will be a conclusion of this thesis consisting in a summary and suggestions for future work.

Chapter 2

Fundamental Concepts and Related Work

To give an overview, this chapter provides an explanation of the different concepts discussed in this thesis, such as facial recognition and its variations as well as optical character recognition. For each of these concepts, state of the art methods or algorithms are briefly explained.

2.1 Short Introduction to Facial Recognition

Already 60 years ago in the 1960s, three programmers, funded by an unnamed intelligence agency, worked on a program that recognizes different faces [15]. Their initial approach involved landmarks that were manually set on the face at locations such as the mouth or the center of the eyes. Distances and distance ratios were then automatically computed and compared to other faces. If the deviations were small enough, a match was made. Later, they searched for ways to automate the setting of landmarks in order to reduce human labor.

Nowadays, face recognition is ubiquitous and unlocking phones with this technology seems to be completely normal for almost everyone. Not only large companies, but also governments invest in facial recognition research but do not publish any numbers on their expenses, due to the fact, that it is up to this day a rather controversial topic. Face++, China's biggest facial recognition company, for example has raised USD 460 million to invest in a face recognition system for a payment software [9]. Information from this and many more companies alike led to an estimated market value of USD 3.86 billion in 2020. This value is expected to grow to USD 8.5 billion by 2025 [25].

2.2 Variations of Face Recognition

There is a large variety of possible application of facial recognition and it can be split into two major categories: Access control and security. Being one of the easiest biometric characteristics that can be used to identify a person, facial recognition got very popular in the last few years and different techniques of recognition have emerged. The term biometrics is composed of two parts, namely the greek words "bios" (life) and "metrikos" (measure). Commonly, facial recognition is divided into the following three categories: "Holistic Matching Methods", "Feature-based (structural) Methods" and "Hybrid Methods". The following sections are dedicated to explain the functionality and usage of each method mentioned.

2.2.1 Holistic Matching Method

The holistic matching approach uses global information of the face, represented by small features that are directly derived from the pixels of the image. With those features, called facial features, a variance can be calculated that is used to differentiate between faces [13]. The earliest successful demonstration of this method is called Eigenfaces. This and a further face recognition technique will be explained in more detail in chapter 2.5.1.

2.2.2 Feature-based (structural) Method

In contrast to the holistic matching approach, the feature based method does not take the whole face region as an input. It extracts only certain regions, called local features, such as the eyes, mouth, and nose and analyzes their location, appearance, and other geometric statistics. The most common method to do so, is to use an edge detection algorithm that, as the name suggests, finds edges, lines, and curves to extract the local features [18]. After the local features are found, a variance is calculated similar to the holistic matching method, which can be used to match the equal faces and distinguish the different ones.

2.2.3 Hybrid Method

In order to combine the advantages of both methods described above, a hybrid approach has been developed. This approach divides the system into the training mode and the classification mode. The training mode is to extract facial features after normalizing the images. Those extracted features are taken as an input for a back-propagation neural network (BPNN) resulting in different scores in the feature space. The classification mode feeds a new picture to the trained BPNN. The closer this score is to one of the faces already in feature space, the higher is the probability, that the faces belong to the same person [14].

2.3 Dimensionality Reduction

In order to convert a picture of a face into a form, that can be analyzed by a machine, faces are represented as points in a high-dimensional image space. This image space has to be reduced in order to minimize complexity and running time of the face recognition without losing too much data. The best known and most used dimensionality reduction methods are called ”Principal Component Analysis” (PCA) and ”Linear Discriminant Analysis” (LDA). A visual representation of both methods is shown in figure 2.1

2.3.1 Principal Component Analysis

The earliest description of a dimenstionality reduction was made by Pearson (1901) and Hotelling (1933). Already 50 years before computers became available to the common people, Pearson statet, that his method of dimentionality reduction ”can be easily applied to numerous problems”. The underlying idea of this method is to use a vector space transformation in order to reduce a large data set with a large number of interrelated variables without losing the important information from the data set. The remaining variables after the transformation were first called ”principal factors” but, to avoid confusion with the mathematical term, were then renamed to the eponymous ”principal components” [12].

2.3.2 Linear Discriminant Analysis

In 1936, another method of dimensionality reduction was developed, the linear discriminant analysis method uses the following three steps. First, they calculate the so called between-class variance, by calculating the separability between different classes. The next step is to calculate the mean and sample distances of each class, called the within-class variance. Finally, a lower dimensional space is constructed in order to maximize the between-class variance and minimize the within-class variance. In other words, the linear discriminant analysis method maximizes the separation distances between different faces and minimizes the distance of faces belonging to the same person [27].

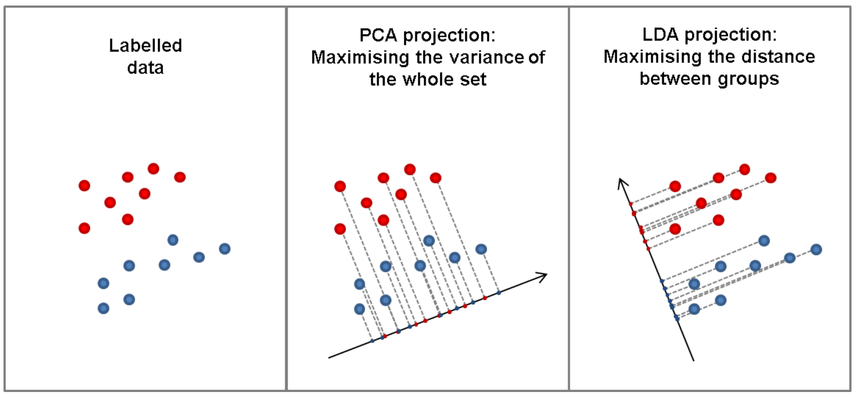

Labelled data
PCA projection: Maximising the variance of the whole set
LDA projection: Maximising the distance between groups

Figure 2.1: Schematic Representation of PCA and LDA [21]

2.4 Optical Character Recognition

Optical Character Recognition (OCR) is a text recognition that can turn text on digital images into ASCII letters. It can not only read printed text, but can also turn handwritten characters into machine-readable text. The recognition is usually done in the following sequence of steps shown in figure 2.2.

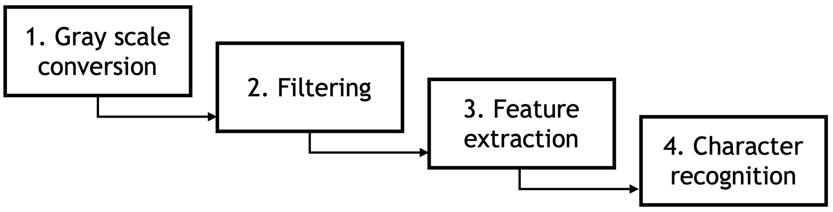


Figure 2.2: OCR Procedure

1. **Conversion into a gray scale image**
   This turns the document into a black and white image where the background and the letters to be read are distinctively different.

2. **Filtering**
   In this step, noise is removed from the image meaning that the edges of black areas will get sharper and the background uniformly white. This filtering is extremely useful when the input images carried much unwanted noise, for example, from a scanner with low resolution or, with respect to this thesis, a picture taken with a laptop camera.

3. **Feature extraction**
   An algorithm searches the black area of the image to find spots that resembles letters or digits. But to do so, it first needs to find the exact boundaries that contain a character. If an input document contains several lines of written text, the rows must be identified. The earlier detected rows have to be split again, such that words are obtained. Now that the words are identified, they have to be split one last time in order to find characters.

4. **Character recognition**
   The last step converts the identified characters into ASCII letters such that it can be read by programs. This can be done with two different methods, feature extraction and pattern recognition, which are described in the next sections.

2.4.1 Feature Extraction

A first approach in character recognition is to decompose letters into features like lines, curves and closed loops. A simple example of this extraction can be seen in figure 2.3 Extracting those feature first, reduces the dimensionality of the representation making it efficient. The extracted features are compared to the dataset of features. If the features match, the character is recognized.

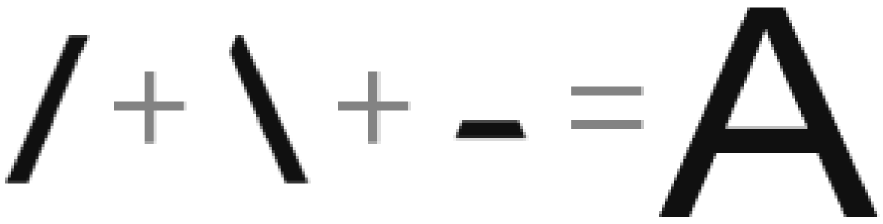


Figure 2.3: Simple representation of feature extraciton [6]

2.4.2 Pattern Recognition

After the characters are extracted from the digital image, the pattern recognition normalizes each by scaling it to 15 x 15 pixels. This new, cropped image can be binarized, meaning that it is turned into a 15 x 15 array containing 0's (representing white area) and 1's (representing black area) [16].

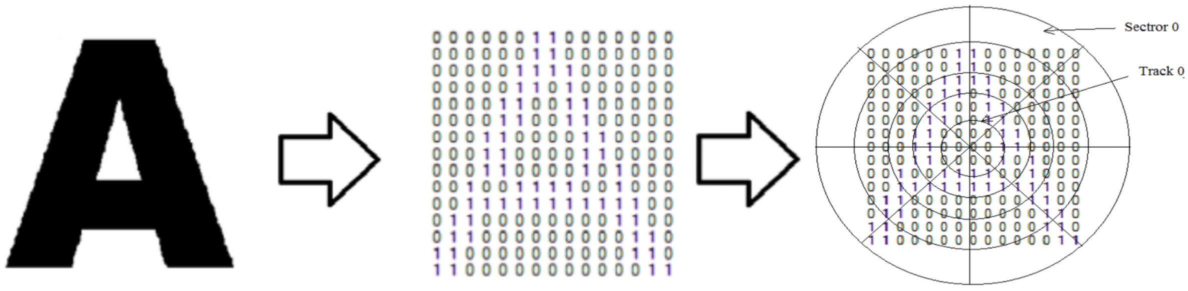

```
0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 0 1 0 0 0 0 0 0
0 0 0 0 1 1 0 0 1 1 0 0 0 0 0
0 0 0 0 1 1 0 0 1 1 0 0 0 0 0
0 0 0 1 1 0 0 0 0 1 1 0 0 0 0
0 0 0 1 1 0 0 0 0 1 1 0 0 0 0
0 0 0 1 1 0 0 0 0 1 0 1 0 0 0
0 0 1 0 0 1 1 1 1 0 0 1 0 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 0 0
0 1 1 0 0 0 0 0 0 0 0 1 1 0 0
0 1 1 0 0 0 0 0 0 0 0 0 1 1 0
1 1 0 0 0 0 0 0 0 0 0 0 1 1 0
1 1 0 0 0 0 0 0 0 0 0 0 0 1 1
```

Figure 2.4: Binarization and division into tracks and sectors [19]

The obtained binarized image is divided into 5 tracks and then subdivided into 8 sectors. This is matched with existing templates and if all parameters match with one template value, the character is identified [19].

2.5 Related Work

This chapter provides an overview of related facial recognition work including common face recognition techniques and existing online student identification tools that use facial recognition.

2.5.1 Common Facial Recogniton Techniques

The most common facial recognition techniques use holistic matching methods. Among them are the eigenface and the fisherface method, which are described in the following subchapters.

2.5.1.1 Eigenfaces

One approach for the detection of human faces is called eigenface. This method solves a two-dimensional recognition problem projecting a face image onto a face space. The combined eigenvectors of a set of faces are called eigenface and define the face space. The training set contains a linear combination of eigenfaces where each face image can be represented. The idea of using eigenfaces came from Sirocich and Kirby [29] when they used principal component analysis to efficiently represent pictures of faces.

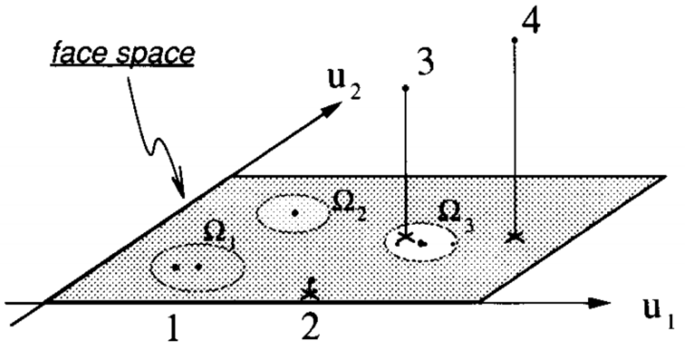


Figure 2.5: Simplified face space with three known faces (Ω, Ω2, and Ω3) and two eigenfaces (u1 and u2) [29].

Typically, images of size 256 x 256 are used for the eigenface method. This will describe a 65'536-dimensional vector. Similar faces are not randomly distributed, but are collected in a smaller subspace called the face space. Each vector in the same face space is a linear combination of the original face image.

2.5.1.2 Fisherfaces

The idea behind fisherface is to use class specific linear methods called Fisher's Linear Discriminant (FLD) for dimensionality reduction. In order to make the model more reliable for classification, the scatter is shaped in such a way that the ratio of between-class scatter and within-class scatter is maximized [3].

2.5.1.3 OpenCV

OpenCV stands for Open Source Computer Vision and is a library created by Intel in 1999. It works on multiple platforms which makes it extremely popular and is focused on real-time image procession. All face recognition methods of OpenCV. In OpenCV, all face recognition models are deduced from a class called FaceRecognizer which is the base class, providing access to different face recognition algorithms. Included are a range of face recognition algorithms like the aforementioned eigenface and fisherface. These recognizers support the following functionalities:

- Training of input pictures
- Prediction of a sample image containing a face
- Loading/saving a trained model
- Setting/getting labels which are stored as a string

These four functionalities are sufficient for most applications that use face recognition. The following method, however, uses a completely different method.

2.5.1.4 DeepFace

A new approach to the conventional machine learning models like ”Principal Component Analyses” or ”Linear Discriminant Analysis” was developed by Facebook AI Research in 2014 and was further refined since. This new approach called DeepFace used deep neural networks and showed improved scaling properties when compared to the conventional methods that had a limited capacity to leverage large volumes of data. According to the paper published by Facebook AI Research, DeepFace reaches an accuracy of 97.35% on the ”Labeled Faces in the Wild” (LFW) dataset, reducing the error of the current state of the art by over 27% and closely approaching a human-level face recognition performance [26]. This dataset, containing 13,233 images of 5,749 people, was specially designed to compare face recognition methods by researchers at the University of Massachusetts [11].

DeepFace has also contributed a significant amount in creating 3D models of faces. This part is crucial for an exact face alignment. The non-planarity of human faces is a challenge to normalize for sufficient face recognition parameters. The face alignment that is used by DeepFace can be divided into two steps: 2D alignment and 3D alignment. The alignment process starts with 6 fiducial points. Those six points are then used to scale and rotate the input image. This approach is often used for conventional face alignment processes. DeepFace also uses the next step in order to align out-of-plane rotations. For this, they use a generic 3D shape model where they wrap around the 2D alignment. Using 67 additional fiducial points, this generates the 3D aligned version of the face.

2.5.2 Optical Character Recognition Methods

The second important building block for this thesis, is the OCR. Therefore, the two most known and widely used methods are described in more detail in the following subchapters.

2.5.2.1 Tesseract

The OCR engine Tesseract is open-source and was developed at HP between 1984 and 1994. It began as a PhD project in software research and had a lead in accuracy, eventhough they did not publish it as an application at first. It used the feature extraction method to recognize letters. Today, Tesseract is maintained by google and it can recognize up to 100 different languages and a large number of fonts, making it a popular OCR engine [23]. For this thesis, the Python version of this engine, PyTesseract, will be used in the evaluation part.

2.5.2.2 PyOCR

Another OCR engine that is easy to use and has some more features than PyTesseract is PyOCR. With this tool wrapper, one can simply change between different OCR tools by changing one line of code.

2.5.3 Existing Work

2.5.3.1 Student Identification Tools using Face Recognition

In order to protect assessment integrity and avoid frauds, student impersonation has to be prevented. To do so, security tools using facial recognition promise schools and universities a safe student verification process. In the course of 2020, software to verify student identification for online assessments, have repeatedly emerged. Table 2.1 provides a description of a variety of such applications.

Table 2.1: Overview of identity verification tools for online exams.

Product	**Description**	**Specifications**	**Cost**
ExamShield [20]	Web portal for online examinations	Live video monitoring, facial recognition	USD 3 per test per student
BioID [4]	Web portal for online examinations	ID photo validation, facial recognition	Cost not specified
ExamSoft [8]	Software for student verification	Face verification using a baseline image taken before examination	Cost not specified
WISE-flow [28]	Web portal for online examinations	Like video monitoring, facial recognition	USD 11 per year per student
SMOWL [24]	Web portal for online examinations	Continuous student authentication with face recognition	Cost not specified

Table 2.1 shows that the market is already filled with applications offering a solution to the online identity verification problem. One disadvantage that these tools have in common is that they are stand-alone products and cannot be used with an existing video conferencing tool. Therefore, the goal of this thesis is to develop a prototype for a cross-tool face verification system that can be used for applications such as Zoom and Microsoft Teams. Another reason for this project is the centralized computing power for face verification at the host. Students do not have to download any programs or deal with new, unfamiliar web-portals. This minimizes the risk of technical obstacles and reduces the stress factor in an already high pressure situation

2.5.3.2 A real-time face recognition system using Eigenfaces

In this work, the author Daniel Georgescu [5] describes the three basic distinction of face recognition. According to him, face recognition can be divided into face verification, face identification, and the watchlist.

1. **Face Verification**
   The first method of face recognition performs a one-to-one search with the intention to compare two pictures and check if the person is the same. This method is used to verify the identity of an individual returning a *True* value if the faces match and a *False* value if the faces are from different people.

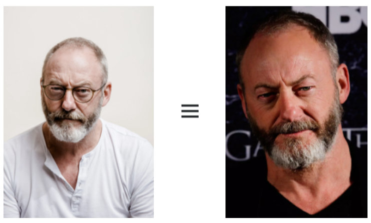


Figure 2.6: Face Verification

2. **Face Identification**
   In this approach, the identity of an individual is not known in advance. The algorithm performs a one-to-many search trying to find the best match within the face database. This method will return the identity of the closest match.

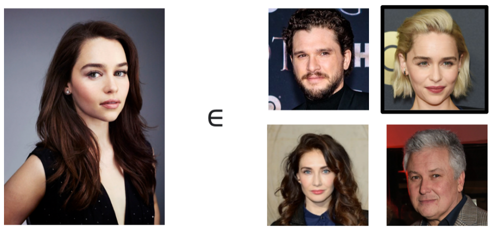


Figure 2.7: Face Identification

3. **Watchlist Task**
   Again, the identity of the individual is not known in the beginning. The aim of this method is to check if the individual is within a list of people. Examples of this task would be comparing flight passengers to a database of terrorists or missing people. The output of this method will be *True* if the face matches an individual within the list and *False* if no match was found.

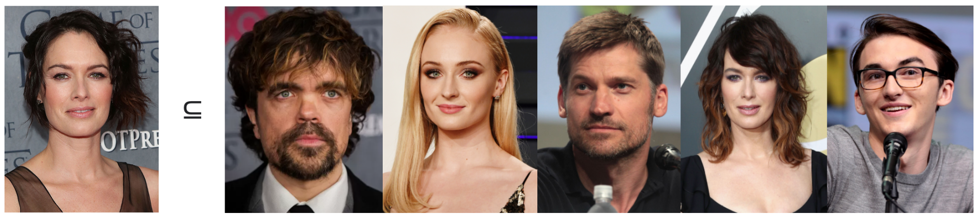


Figure 2.8: Watchlist Task

To find a face on the image, Georgescu uses the Canny edge detection algorithms as a means of noise reduction and the Viola Jones face detection algorithm to locate the face on the filtered picture.

Chapter 3

Prototype Design and Implementation

Part of this thesis consists of creating a working prototype that can recognize individuals in an online video conference, e.g. students in an online examination. This chapter provides an overview of the planing and the implementation of the prototype, starting with the definition of the requirements. After a successful requirement specification, the design of the prototype can be elaborated. Once the requirements are set and the design is defined, the implementation will be explained in detail using code snippets for clarification.

3.1 Requirements

The definition of requirements can be derived from the goals of this thesis, namely to create a prototype of a software, that automates identity verification in an online environment using face recognition. One application of this software is the identification and verification of students in an online examination. The requirements for this application can be divided into user requirement specification and design requirement specification, allowing a separation of functional and non-functional requirements.

3.1.1 User Requirement Specification

The prerequisites that must be specified first are the user requirement specification, or short URS. As the name suggests, they examine the requirements from the user's perspective, meaning that they define the functionalities of the application. The most relevant user requirements for the prototype are presented in the list below.

- **Running on one machine**
  Only the examination caretaker should run this application that requires access to the university database. This preserves from unnecessary risks of technical difficulties and the program does not need to be installed by every student.

- **Independent from one online video conferencing tools**
  A cross platform solution solves the problem of using various online video conferencing tools for different lectures.

- **Verifying multiple faces simultaneously**
  Simultaneous face verification saves time and there is no need for a manual switching between the faces.

- **Intuitive real-time visual feedback**
  A constant visual feedback gives the exam caretaker an insight in the recognition task. This feedback needs to be intuitive in a way, that the user can detect suspicious incidents.

3.1.2 Design Requirement Specification

After the user specifications have determined what the application shall do, the design requirement specifications (DRS) need to examine how such shall function in more technical detail. Among other things, these contain demands on the computational cost and the usage of a screen capture method instead of a webcam-based system. These demands are design-relevant and have an influence on the architecture of the prototype. A compilation of the design requirement specifications is listed below.

- **Accurate face detection.**
  The first design requirement consists in an accurate face recognition. This ensures correct identification and verification of students.

- **Low computational cost.**
  This application should be able to run on an average laptop and does not need an expensive high performance computer. Lecturers should be able to use their own laptop at a location of their choice.

- **Real-time facial recognition.**
  This allows the program to constantly respond to changes and deliver a corresponding live feedback.

- **Use screen capture instead of video identification.**
  Taking the screen content as the input source solves many issues. The software only needs to run on one machine and is independent from camera systems and video conferencing tools.

Implementing these requirements, a system can be built that meets the challenges for an individual authentication using facial recognition.

3.2 Prototype Design

Once the user and design requirements have been specified, a design can be elaborated. The activity diagram offers a way to structurally represent the processes and behaviors of the application. This provides a clear demonstration of the logic behind the application and simplifies the description of the steps. Figure 3.1 shows the activity diagram and the enumerated list below serves as a step-by-step description of each activity.

1. **Run Face Training with Database Pictures.**
   Once the program is started, it will start an algorithm that learns the faces from the simulated university database. However, this only needs to be done once per class, unless new changes are made in the database.

2. **Capture Screen Image.**
   At this point, the screenshot content needs to be capured and saved as an image. To do so, screen capture methods are used.

3. **Detect Faces on Image.**
   Here, an algorithm detects faces on an image. This serves two purposes. On the one hand, it is important to know the number of faces on the image, to later create the correct amount of frames on the screen. on the other hand, the exact coordinates of the found faces are obtained. These are used in the next step.

4. **Search Matching Face in Trained Faces.**
   In this part, the faces found in step 3 are matched one by one with those in the database. Thus, a True or False is returned for each face.

5. **Found Matching Face?**
   This part is a decision point. The further procedure of the application is determined by the output. If the face is found in the database, step 7 is executed. However, if the face is unknown, step 6 follows.

6. **Create Red Frame around Face.**
   If the face is not found in the database, a red semi-transparent window will appear on the user's screen, framing the unrecognized face. The window will be labeled as ”unknown”.

7. **Create Green Frame around Face with Name Tag.**
   If the face is detected, the frame will be green-colored and the label corresponds to the name of the found individual.

8. **Add Person to the Attendance List.**
   The recognized person is added to a list that memorizes the present individuals. This list will be used for output in step 10.

9. **Terminate Program?**
   At this point it is checked whether the user has terminated the program or whether it should continue to run. If no user input is received, the program jumps to step 2 and starts the loop again. If the user has terminated the program, step 10 is executed.

10. **Create Output File.**
    In this step, the output file is updated using the list of present students. When this happens, the application adds a column to the attendance list, which is given as an input to the applicaiton. To do so, it adds the current date on the top row and adds the value ”x” for each student that was found during runtime. This will create an attendance list that can be used over a longer period of time, e.g. a semester. After that, the program will be terminated.

The following page shows the activity diagram of the prototype with the enumeration going from 1 to 10 as described above.

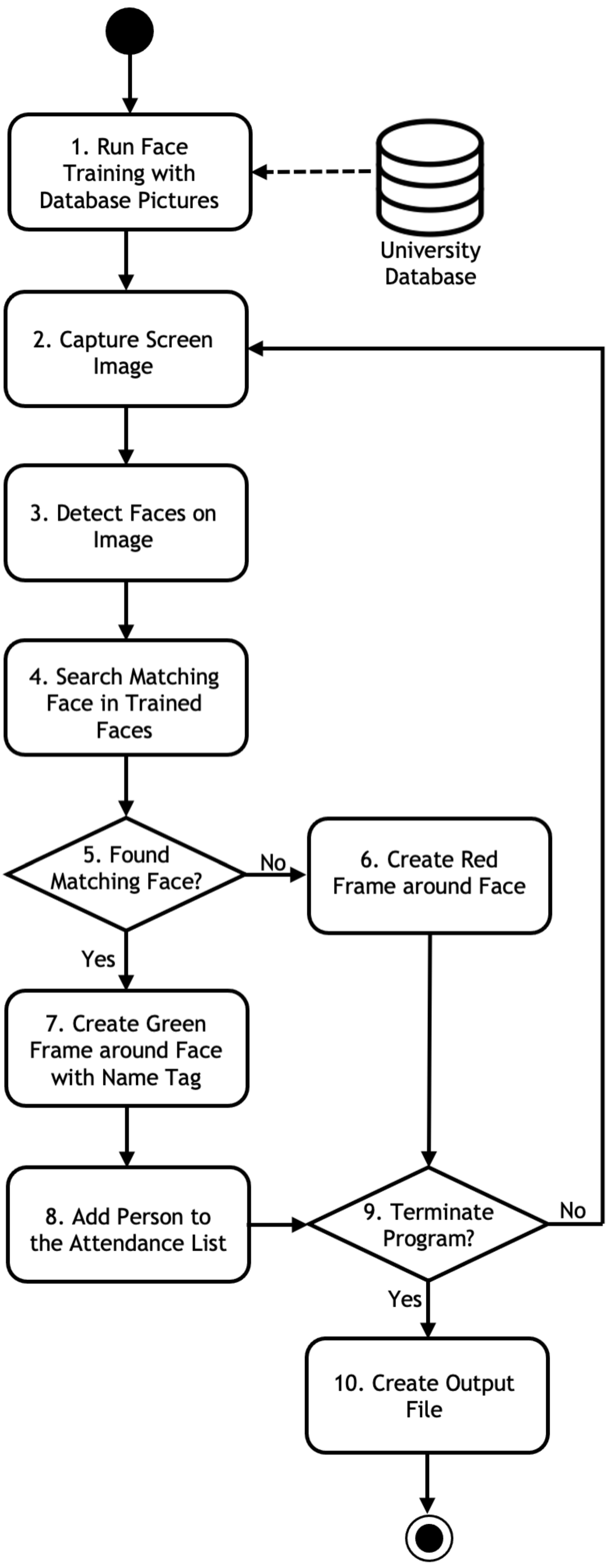

1. Run Face Training with Database Pictures
University Database
2. Capture Screen Image
3. Detect Faces on Image
4. Search Matching Face in Trained Faces
5. Found Matching Face?
No
6. Create Red Frame around Face
Yes
7. Create Green Frame around Face with Name Tag
8. Add Person to the Attendance List
9. Terminate Program?
No
Yes
10. Create Output File

Figure 3.1: Activity Diagram for the Prototype Process

3.3 Implementation

To gain an understanding of the code, this chapter elucidates the implementation in more detail. The code can be divided into three basic sections. It starts with the initialization including the creation of the face encodings and the windows. The second section contains the main loop which is responsible for face recognition and correct display of the colored frames. As soon as the user interrupts the loop, the code enters the terminal phase. In this section, the output file, i.e. the attendance list, is updated. The exact functionalities of the three sections together with code snippets are described in the following chapters starting with the input specifications.

3.3.1 Input Specification

Since this thesis implements a prototype and not the final application connected to the university database, the input can be simulated as follows:

1. **Student Pictures**
   The university database is simulated by a folder. This represents a class containing the student images of each enrolled student. The names of the image files must correspond to the name of the student.

2. **Attendance List**
   The attendance list serves the user as an output and will be updated after every run of the program. An Example of an attendance list can be found in figure 3.2. An empty attendance list contains the student names in the first column. After every run, the application adds a new column with the date on the first line and an ”x” at each present student.

3.3.2 Code Setup

As soon as the application starts, a running variable is set. As the name indicates, this is *True* as long as the main loop should be executed. Next, a dictionary is created to register the student presence of this run. The current date will be added to the dictionary and will be used when generating the output.

```
        running = True
        attendance_list = {"date": datetime.now().strftime("%d/%
            m/%Y")}
```

To be able to terminate the main loop at any time by a key press, a new thread called *interrupt* is started in parallel. This thread calls the function get_input which awaits user input.

```
        interrupt = threading.Thread(target=get_input)
        interrupt.start()
```

Furthermore, the resolution of the monitor must be defined to ensure that the screenshots cover the correct area.

```
        mon = {'top': 0, 'left': 0, 'width': 1640, 'height':
            950}
        screenshot = mss.mss()
```

To find faces, a special method called Cascade Classifiers must be defined. The code to find faces frontally looks as follows:

```
        classifier = 'cascades/data/
            haarcascade_frontalface_default.xml'
        face_cascade = cv2.CascadeClassifier(classifier)
```

The last step of the initialization consists in setting up the colored frames. For this purpose, a main window is created, which has 0 opacity and is only 1 x 1 pixel in size. This window, also called root window, is only used to create or remove child windows during the main loop more easily and should be invisible to the user.

```
        # Initiate the root window
        root = Tk()
        # Set root title to "Main"
        root.title("Main")
        # Set visibility & coordinates
        root.overrideredirect(True)
        root.wait_visibility(root)
        root.wm_attributes('-alpha', 0.0)
        root.geometry('%dx%d+%d+%d' % (width, height, x, y))
        # Set the window on top of every other window
        root.attributes('-topmost', True)
        root.lift()
        # Create empty list for windows and labels
        windows = []
        labels = []
```

After this section, the main loop is executed. As long as the *running* variable is set to *True*, the code, explained in the following section, is executed.

3.3.3 Main Loop

At this point, the program enters the Main loop. First, a screenshot is taken. However, if there are windows in front of a face, this could lead to a distortion of the image. For this reason the opacity of all visible windows is set to 0. After a screenshot is taken, the windows become visible again. This happens within milliseconds and is undetectable to the human eye.

```
    for win in windows:
        # Set Window Opacity to 0
        win.wm_attributes('-alpha', 0.0)
    img = screenshot.grab(mon)
    for win in windows:
        # Set Window Opacity to 0.2 (overlay)
        win.wm_attributes('-alpha', 0.2)
```

Next, the image is processed. To do this, it must first be converted into a numpy array. The method used here to take a screenshot converts the image into an rgba image. In order to format the image into a grayscale image, this must first be an rgb image. For this the function *rgba2rgb* is used here. After that the image can be converted as described. The resulting grayscaled image serves as the input of the face detection.

```
frame = np.array(img)
frame = rgba2rgb(frame)
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, scaleFactor
    =1.05, minNeighbors=5, minSize=(80, 80))
```

The list *"faces"* contains the following data for each detected face: x and y coordinates, width, and height of the face. Now that the number of faces is known, it is checked whether the needed amount of windows is currently present. If there are more faces on the screenshot than there are windows, new windows must be added and if there are fewer, some must be deleted.

Once the number of windows that needs to be created is correct, each face is analyzed in turn.

```
count = 0
for (x, y, w, h) in faces:
    curWin = windows[count]
    # Adjustment of Labels
    if count < len(labels):
        curLab = labels[count]
    else:
        curLab = Label(curWin)
    labels.append(curLab)
```

The next step is a central element of this thesis. The face that is currently in focus is compared with the images in the simulated university database. For this purpose, the DeepFace library using the "Facenet" model is used. An evaluation of the available DeepFace models can be seen in chapter 4.

```
        result = DeepFace.find(roi_face, db_path=picture_path,
           model_name="Facenet", enforce_detection = False)
```

The *result* is a numpy array where if *result.shape[0] > 0*, a match has been found. If this is the case, the window color, that is set to red by default, will turn green. Additionally, the name of the detected student will be added to the dictionary of the attendance list. If the student already has an entry in the attendance list, the corresponding counter increases by one. This serves to recognize false matches that can be excluded by setting a threshold value.

```
        col = "red"
        if result.shape[0] > 0:
            # Get student name
            student_name = result.iloc[0].identity.split('/')
               [-2]
            # Set color to green
            col = "green"
            # Update the attendance list
            if student_name not in attendance_list:
                attendance_list[student_name] = 0
            elif attendance_list[student_name]<20:
                attendance_list[student_name] += 1
        else:
            # Set the label to "unknown"
            student_name = "unknown"
```

However, if the face is not found in the simulated university database, the window color remains red and the *student_name* is set to "unknown". In the last step, the *student_name* is set as a label for the window and the window color is set accordingly.

```
        # Set the labels
        curLab.configure(text=student_name)
        curLab.pack()
        # Set visibility and update the coordinates
        curWin.wm_attributes('-alpha', 0.2)
        curWin.geometry('%dx%d+%d+%d' % (w / 2, h / 2, x / 2, y
           / 2))
        curWin.configure(bg=col)
        curWin.update()
        count += 1
```

After the program iterated through all faces and adjusted the windows, the running variable will be checked. If running is still True, the loop will start again from the beginning. However, if the user has set running to False by a keypress input, the next step will follow.

3.3.4 Generate Output

As soon as the user stops the program by pressing "s", the end phase of the program is initiated. First, the global variable *running* is set to *False* causing the main loop to stop.

```
    global running, attendance_list
    running = False
```

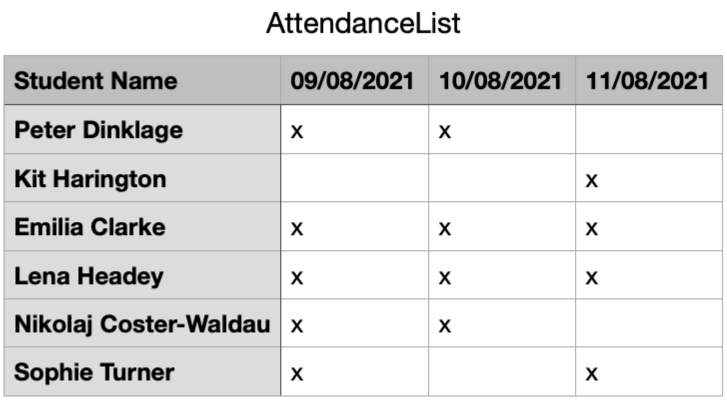
After that, the attendance list has to be updated. For this purpose, the current attendance list must be read. To do this, the code below can be used. First, an empty string is created which is used to copy the current document and add new information at the end of each row. Then, the file is opened and read line by line. On the first row, the date is added. After that, it is checked if the student with the given name was in the dictionary of the present individuals. Additionally, a threshold can be set here, which in this case was set to 3. This means that if an individual was recognized less than three times, he/she is not considered present.

```
to_write = ""
with open("AttendanceList.csv") as list:
count = 0
lines = list.readlines()
for line in lines:
    line = line.rstrip()
    line += ";"
    # Add the date in the first row
    if count == 0:
        line += str(attendance_list["date"])
    count += 1
    name = line.rstrip().split(";")[0]
    # If the person was present & over the threshold
    threshold = 3
    if name in attendance_list and attendance_list[name] >=
        threshold:
        line += "x"
    to_write += line+"\n"
```

The string *to_write* recreates the attendance list and adds the new information such that the attendance list can be updated by writing said string in the file.

```
with open("AttendanceList.csv", "w") as writer:
    writer.write(to_write)
```

The attendance list is saved as a csv and can be opened and viewed with excel. The following figure shows an example of an attendance list. The advantage of this representation is that it can be used over the course of an entire semester. A lecturer can run this program to check the attendance for both, lectures and online examinations.


AttendanceList

Student Name	09/08/2021	10/08/2021	11/08/2021
Peter Dinklage	x	x	
Kit Harington			x
Emilia Clarke	x	x	x
Lena Headey	x	x	x
Nikolaj Coster-Waldau	x	x	
Sophie Turner	x		x

Figure 3.2: Example of an Attendance List

The attendance of a student is represented by an ”x” at the corresponding date. This is a simple representation, as it is often used to check presences. Having this automated, saves manual work, valuable time, and is not prone to human error.

3.4 Algorithm for Method Comparison

In this chapter, the design and the implementation of the DeepFace comparison method will be explained in more detail.

The comparison method uses the following input:

- ”models” - A list containing all models that shall be compared
- ”pictures” - A list of pictures. One picture per person to learn the faces
- ”folders” - A list of folders. Each folder is named after a person that should be compared and contains three pictures of given person.
- ”picture_location” and ”folder_location” - Location of folders and pictures

3.4.0.1 Code for DeepFace Comparison Method

This testing follows a sequential set of steps shown in figure 3.3.

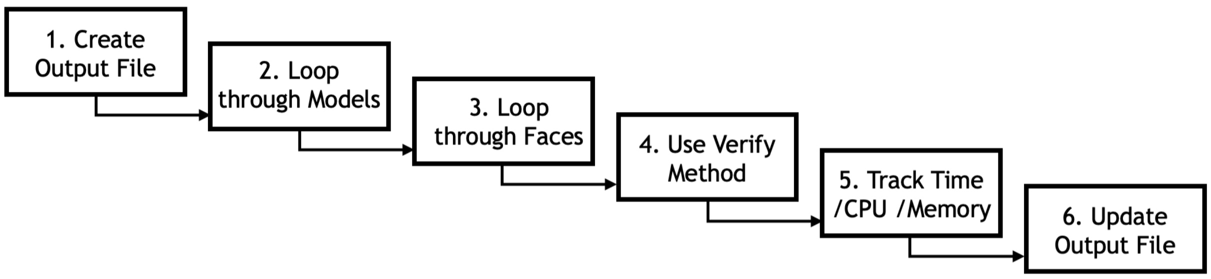


Figure 3.3: DeepFace Comparison Procedure

1. The text document *”TestResults.txt”* is created, which can later be easily processed as csv via excel. This document serves as the output and stores the resulting *True/-False* values of the face recognition.

```
# Open / create Textfile for the results
 with open('TestResults.txt', 'w') as results:
```

2. In order to run this method for all available DeepFace models, a for loop will iterate over a list containing all models.

```
    # Loop over the Models
    for model in models:
```

3. similar to the models, a loop is needed to iterate over each reference picture.

```
        # Loop over each Person
        for person in pictures:
```

4. Before the verify method can be called, a string has to be defined, that stores the *True* values as 1's and the *False* values as 0's. Further, counters will be set, that are needed to analyze the models in terms of time, CPU, and memory usage.

```python
            # String that will be filled with 0's (False) and 1's (True)
            truefalse = ""
            # Floats counting time / CPU / memory
            total_time = 0.0
            total_cpu = 0.0
            total_memory = 0.0
```

The following code snippet shows the iteration over every file inside *"folders"*. The verify method checks for each file, which is not hidden, if it matches the person. If a match is made, it adds a 1 to the *"truefalse"* string, else a 0.

```python
            for folder in folders:
                # Walk through the each folder
                files = list(os.walk(folder_location + folder))[0][2]
                for file in files:
                    # Ignore the hidden files containing "DS_Store"
                    if "DS_Store" in file:
                        continue
                    # Save current time in a variable
                    date = datetime.now()
                    # Compare the current face with the picture using given model
                    result = DeepFace.verify(picture_location + person, folder_location + folder + "/" + file, model_name=model, enforce_detection=False)
                    # Add a 1 to truefalse if the face is the same, else add 0
                    if result["verified"]:
                        truefalse += "1,"
                    else:
                        truefalse += "0,"
```

5. In order to capture the time, cpu, and memory usage correctly, it has to be added after each iteration. This will produce an accurate average in the end.

```
                # Adding time / cpu / memory
                total_time += float(str(datetime.now() - date)[5:])
                total_cpu += psutil.cpu_percent()
                total_memory += psutil.virtual_memory()[2]
        # Calculate averages
        average_time = total_time / (len(folders)*len(folders[0]))
        average_cpu = total_cpu/(len(folders)*len(folders[0]))
        average_memory = total_memory / (len(folders)*len(folders[0]))
```

6. In the last part, the output is prepared. For this purpose, a string is filled with the retrieved information and is written into the output file. The separation by commas allows the document to be opened as csv via excel.

```
        # Prepare the Output
        out = truefalse+str(total_time)+",,"
            +str(average_time)+",,"
            +str(average_cpu)+",,"
            +str(average_memory)
        # Print Output to the TestResults File
        print(out, file=results)
```

Chapter 4

Evaluation

This chapter describes the evaluation and analysis of the prototype concerning the different deployed methods. Many different approaches, which are described and compared in detail in the following sections, were taken into consideration to find a fitting algorithm. The subsequent aspects are the cornerstones of this evaluation.

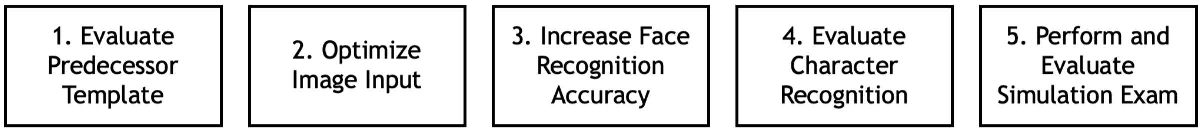

1. Evaluate Predecessor Template
2. Optimize Image Input
3. Increase Face Recognition Accuracy
4. Evaluate Character Recognition
5. Perform and Evaluate Simulation Exam

Figure 4.1: Aspects of Evaluation

Both the results and the procedure of the evaluation are explained in detail in the following chapters.

4.1 Face Recognition Template using OpenCV

To start this evaluation the OpenCV template, processing camera input to compare to a set of pre-learned pictures, was followed. This method was also used by the predecessor of this thesis [10], that studied more closely the positive and negative aspects of this approach. One drawback of this method is that the camera is used by both, the video conferencing tool and the identity verification application. Another drawback is the lack of a sufficiently large learning set. Each student has only a single photo in the database, which is too few to learn from for an accurate face recognition using the OpenCV-template. This lead to the recommendation to use screenshots to counteract those negative aspects. That not only solves the problem with the camera being already used for the conferencing tool, but also would allow to use several screenshots to increase the size of the learning set. Thus, the student picture located in the university database serves as the comparison picture. Another advantage that this method holds is that it runs on a single host centralizing the computing power and avoiding additional tasks for students.

To collect screenshots, the python library called PyAutoGUI [1] was selected, which is specialized to automate graphical user interfaces. It is straightforward to use and according to the official documentation, the *pyautogui.screenshot()* function takes about 0.1 seconds on a 1920 x 1080 screen.

Running the OpenVC template with PyAutoGUI proved to be very slow, in which a single frame took 0.43 seconds to take, meaning that if 10 pictures were needed for an accurate face recognition, gathering the frames would take over 4 seconds (data collected based on a 2016 MacBook Pro with the 2.7 GHz quad-core Intel Core i7 processor). This finding initiated a process of method comparisons, that shaped the nature of this thesis into a comparative one. The following subsections explains the exact comparisons that were conducted for the evaluation. At the end of each section, the decision for the further procedure is explained and justified.

4.2 Increase Frame Rate of the Screen Capture

The OpenCV approach, described in the previous chapter, take screenshot input in order to learn recognizing the given person. Valuable time can be saved by increasing the frame rate of the screen capturing method. For that, three of the supposedly fastest screen capture methods for Python have been compared in this section: PyAutoGUI, PIL ImageGrab, and Python MSS.

- As shortly introduced in the previous chapter, PyAutoGUI is a Python library that was made to easily automate graphical user interfaces. To do so, it contains a method to capture screenshots. To define the screenshot region, the variable *region* can be changed using four parameters *left, top, width, height.*

- PIL, short for Pillow, is a Python Imaging Library that adds image editing capabilities to the Python interpreter. The ImageGrab module from Pillow is a method to copy contents of the screen directly to the clipboard or save it in a file. By default, the entire screen will be captured. To change this, one can add the variable *bbox* with the parameters *X_Left, Y_Top, X_Right, Y_Down.*

- Python MSS was developed as an ultra fast cross-platform module that can take multiple screenshots. It can be easily embedded into the program and can save a screenshot directly as a PNG file and uses a dictionary, called monitor, to define the screenshot region using *left, top, width, height.*

To decide which of the mentioned methods is the most suitable for this purpose, a performance test was carried out. The test shall compare running time, CPU consumption and memory usage. The results of this test file are shown in table 4.1. It is is to be mentioned that the table shows the average values of a total of 10 screenshots taken.

Table 4.1: Results of Performance Tests

Method	**Time** [s]	**CPU** [%]	**Memory** [%]	**Method Call**
PyAutoGUI	0.58 s	16.53 %	42.42 %	pyautogui.screenshot(region)
PIL ImageGrab	0.34 s	14.31 %	42.6 %	ImageGrab.grab(bbox)
Python MSS	0.06 s	14.93 %	43.01 %	screenshot.grab(monitor)

The second column of the table shows the time that was needed to capture one screenshot in seconds. For the CPU and the memory consumption, displayed in the third and fourth column, a cross-platform library called ”psutil” was used [22]. Psutil stands for ”Python System and Process Utilities” and can retrieve performance information on running processes. The function *cpu_percent* that was used to get the CPU usage in percent. The second psutil method, *virtual_memory*, returs statistics on the memory usage. This list includes following types of memory data: total, avalilable, percentage, used, free, active, inactive and more. From the many types of memory data the percentage is most significant in this context and it is calculated as follows: *(total - available) / total * 100.*

The parameters x, y, w and h, that were used for the screenshot size are shown here for reasons of transparency and replicability. It is also worth mentioning that these tests were run on a 2016 MacBook Pro with the 2.7 GHz quad-core Intel Core i7 processor.

- Left side x = 0
- Top side y = 0
- Screen width w = 1640
- Screen height h = 950

Each of the three methods have their pros and cons. Python MSS is by far the fastest of the evaluated methods being over 5 times faster than second place. In terms of CPU percentage is PIL ImageGrab in the lead, followed closely by MSS and PyAutoGui. The first place in memory consumption is PyAutoGUI, just a little less memory intensive than the other two. In summary, only minor differences can be seen in the CPU and memory usage. The most significant difference between the methods lies in the running time, where Python MSS is by far superior. As a result, said method will be used at the testing stage and in the final prototype.

4.3 Higher Accuracy of Face Recognition

Once a satisfactory solution for the input has been found, the accuracy of facial recognition was tested and compared with alternatives. For this purpose, the OpenCV template was used to obtain an initial value that can be used to compare to other face recognition methods. In order to obtain such a value, the OpenCV template is provided test data and evaluated on accuracy. The dataset, created specifically for these evaluations, consists of 15 cast members of the popular television series ”Game of Thrones”.

The results of the test are shown in the table 4.2. The first row contains names of people that the program tries to recognize. Each following row shows one test run where the program tries to assign the face of a person the left to one of the people on the top line. E.g. for the first test run it means that the program gets three pictures of Alfie to learn from but assigns it incorrectly to Conleth. A perfect run would thus only have X’s on the diagonal, which is marked with the dashes. On the last column is the confidence value, which indicates the distance to the closest item in the database. A confidence value of 0 would be a perfect match.

Table 4.2: Evaluation of the Facial Recognition using 3 reference Pictures

OpenCV	Alfie	Carice	Conleth	Emilia	Gwendoline	Iain	Isaac	Kit	Lena	Liam	Maisie	Nethalie	Nikolaj	Peter	Sophie	Confidence
Alfie	-	0	X	0	0	0	0	0	0	0	0	0	0	0	0	112
Carice	0	-	X	0	0	0	0	0	0	0	0	0	0	0	0	106
Conleth	X	0	-	0	0	0	0	0	0	0	0	0	0	0	0	104
Emilia	0	0	X	-	0	0	0	0	0	0	0	0	0	0	0	85
Gwendoline	0	0	X	0	-	0	0	0	0	0	0	0	0	0	0	98
Iain	0	0	X	0	0	-	0	0	0	0	0	0	0	0	0	116
Isaac	0	0	0	0	0	0	-	X	0	0	0	0	0	0	0	154
Kit	0	0	X	0	0	0	0	-	0	0	0	0	0	0	0	70
Lena	0	0	X	0	0	0	0	0	-	0	0	0	0	0	0	113
Liam	X	0	0	0	0	0	0	0	0	-	0	0	0	0	0	103
Maisie	X	0	0	0	0	0	0	0	0	0	-	0	0	0	0	126
Nethalie	0	0	X	0	0	0	0	0	0	0	0	-	0	0	0	72
Nikolaj	0	0	X	0	0	0	0	0	0	0	0	0	-	0	0	80
Peter	0	0	X	0	0	0	0	0	0	0	0	0	0	-	0	68
Sophie	0	0	0	0	0	0	0	X	0	0	0	0	0	0	-	96

This table shows that the OpenCV method, where three pictures of each person is given as a training set, behaves in an unexpected way. There seems to be an extreme bias towards one person, in this case towards ”Conleth” with a few outliners towards ”Alfie” and ”Kit”. A short research confirms that the OpenCV face recognition needs more than three pictures for better results. Ideally there should be 10 or more pictures of each person [7]. Optionally, one can also add an ”Unknown” dataset filled with different people that are not within the other pictures.

In order to test this with more picture, a larger database with 10 pictures per person was created and tested similar to the previous trial.

Table 4.3: Evaluation of the Facial Recognition using 10 reference Pictures

OpenCV	Alfie	Carice	Conleth	Emilia	Gwendoline	Iain	Isaac	Kit	Lena	Liam	Maisie	Nethalie	Nikolaj	Peter	Sophie	Confidence
Alfie	-	0	0	0	0	0	0	0	0	X	0	0	0	0	0	58
Carice	0	-	0	0	0	0	0	0	0	X	0	0	0	0	0	40
Conleth	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0	41
Emilia	0	0	0	-	0	0	X	0	0	0	0	0	0	0	0	35
Gwendoline	0	0	0	0	-	0	X	0	0	0	0	0	0	0	0	50
Iain	0	0	0	0	0	-	0	0	0	0	0	X	0	0	0	64
Isaac	0	0	0	0	0	0	-	0	0	X	0	0	0	0	0	85
Kit	0	0	0	0	0	X	0	-	0	0	0	0	0	0	0	23
Lena	0	0	0	0	0	0	X	0	-	0	0	0	0	0	0	63
Liam	0	0	0	0	0	0	0	0	0	X	0	0	0	0	0	46
Maisie	0	0	0	0	X	0	0	0	0	0	-	0	0	0	0	79
Nethalie	0	0	0	0	0	0	X	0	0	0	0	-	0	0	0	23
Nikolaj	0	0	0	0	0	0	0	0	0	0	0	X	-	0	0	30
Peter	0	0	X	0	0	0	0	0	0	0	0	0	0	-	0	21
Sophie	0	0	0	0	0	0	X	0	0	0	0	0	0	0	X	18

Table 4.3 that the accuracy of the OpenCV method has increased but is still far from a reliable solution. With only 3 matches out of 15 attempts, the accuracy is at a low 20%. Another difference compared with the experiment with 3 learning images is that the confidence values became smaller. The first test had an average confidence value of 100.2 whereas this test run has an average confidence value of 45. This findings has brought this thesis to an important decision with the following options:

1. Using the OpenCV method with a high amount of pictures for the training set.
2. Searching methods with a higher accuracy that need fewer pictures.

The high number of pictures required by the OpenCV method leads to a longer duration of capturing screenshots. In an online examination the organizer would have to wait several seconds for each student until sufficient pictures are taken. Only after the pictures are captured, the program can start learning the faces and compare them to the student foto in the database of the university. Finding better methods that require fewer pictures in the learning dataset could save valuable time. Furthermore, if there exists a method, that only needs a single picture for an accurate face recognition, the learning could be done at the start of the program and verify the students in real time. Such a method is offered by the DeepFace library with its modules specialized in face recognition even with only a few comparison images.The following chapter discusses the DeepFace modules mentioned and the tests that were performed as part of this thesis.

4.3.1 Face Recognition Alternative using DeepFace

Facebooks DeepFace offers an alternative to the OpenCV approach that not only uses fewer pictures but also promises a higher accuracy. According to Facebook AI Research, DeepFace can reach an accuracy of 97.35% which is almost on a human level. It also offers state of the art models for the analysis like VGG-Face and Google Facenet. Each method and their according Accuracy on the popular LFW (Labeled Faces in the Wild) dataset [11] is shown in the table below.

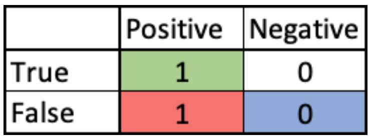
Table 4.4: DeepFace Models Overview

Model	**Developer**	**LFW Accuracy**
VGG-Face	University of Oxford	97.78%
FaceNet	Google	99.63%
OpenFace	Carnegie Mellon University	93.80%
DeepFace	Facebook	97.35%
DeepID	Chinese University of Hong Kong	99.15%
ArcFace	Imperial College London	99.40%

According to Table 4.4, FaceNet and ArcFace show the best results with 99.63% and 99.40% respectively. To check whether these data were accurate, they were compared to the same dataset as from the first tests. Earlier three reference pictures were used to recognize one face. However, to evaluate the DeepFace library, only one reference picture is used but is checked against three faces each. This test is done for each of the DeepFace Methods mentioned in Table 4.4.

4.3.2 Comparing DeepFace Models

The results of the tests, for which the code was shown in **chapter 3.4.0.1**, is examined in detail in this section. The output of the algorithm is a CSV file, which after a short processing looks like Table 4.5. For a better visualization, the results were colored. The green cells show that the model correctly recognized the reference picture. For example, VGG Face recognized two out of three images of ”Alfie”. The blue cells indicate false negatives which means that a face corresponding to a reference picture was not recognized but should have. The red ones are false positives. Here, a face was allocated to the wrong reference picture.



	Positive	Negative
True	1	0
False	1	0

Figure 4.2: Confusion Matrix

Table 4.5: Comparison of DeepFace Models

DEEPFACE MODEL		Alfie			Carice			Conleth			Emilia			Gwendoline			Iain			Isaac			Kit			Lena			Liam			Maisie			Nethalie			Nikolaj			Peter			Sophie		
VGG-Face	Alfie	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Carice	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Conleth	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Emilia	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
	Gwendoline	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Iain	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0
	Isaac	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Kit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Lena	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Liam	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Maisie	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	Nethalie	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
	Nikolaj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
	Peter	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0
	Sophie	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
Facenet	Alfie	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Carice	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Conleth	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Emilia	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Gwendoline	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Iain	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Isaac	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Kit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Lena	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Liam	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Maisie	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	Nethalie	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
	Nikolaj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
	Peter	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0
	Sophie	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
OpenFace	Alfie	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Carice	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Conleth	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Emilia	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	Gwendoline	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Iain	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Isaac	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	Kit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Lena	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Liam	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Maisie	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Nethalie	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
	Nikolaj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Peter	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Sophie	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Deepface	Alfie	1	0	0	0	0	1	0	1	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	1	0	0	0	1	1	1
	Carice	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1	1	1	0	1	0	0	0	0	0	0	1
	Conleth	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Emilia	1	0	0	0	1	1	1	1	0	0	1	0	1	1	1	0	0	0	1	0	0	0	1	1	1	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	1	1	1
	Gwendoline	1	0	0	0	0	1	0	1	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	1	0	0	0	1	1	1
	Iain	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Isaac	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Kit	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Lena	1	0	0	0	1	1	1	0	0	0	1	0	1	0	1	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	1
	Liam	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Maisie	0	0	0	1	0	1	0	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	1	0	1	1	0	0	0	1	0	0	0	1	1	1
	Nethalie	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	1
	Nikolaj	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Peter	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	Sophie	1	0	0	1	0	1	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	1	1
DeepID	Alfie	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	1	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	1	0	1	1	0	0	0	0	0	1	0
	Carice	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
	Conleth	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Emilia	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	1	1
	Gwendoline	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	Iain	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Isaac	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Kit	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Lena	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
	Liam	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Maisie	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Nethalie	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
	Nikolaj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Peter	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Sophie	0	0	0	1	0	0	1	1	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
ArcFace	Alfie	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Carice	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Conleth	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Emilia	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Gwendoline	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Iain	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
	Isaac	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Kit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Lena	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Liam	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Maisie	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	Nethalie	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
	Nikolaj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
	Peter	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0
	Sophie	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Table 4.6 shows that the models VGG Face, Facenet and ArcFace deliver the best results. With only 5 false negatives, Facenet leads in terms of accuracy. ArcFace has 4 false negatives, one fewer than Facenet, but is has 5 false positives, which puts this method in second place. With only three false negatives, VGG Face has the highest number of correct matched faces. However, it also has 28 false positives, which is high compared to the previously mentioned methods. The other three methods do not show promising results for face recognition using only a single picture for reference. While OpenFace makes few guesses and thus has a high number of wrong negatives, DeepFace and DeepID tend to have the issue that these methods make too many guesses. This leads to an increased number of wrong positives for these methods.

Table 4.6: Error Analysis of DeepFace Models

DeepFace Model	**False Positives**	**False Negatives**	**Total Error**
VGG-Face	28	0	28
Facenet	0	5	5
OpenFace	2	37	39
Deepface	97	21	118
DeepID	43	39	82
ArcFace	5	4	9

For each model, the average Time, CPU usage and memory usage have been recorded and are listet in table 4.7. It allows a more computational comparison between the models. In terms of time needed for the face recognition, DeepID is with under 0.7 seconds per comparison the fastest method, followed by OpenFace with takes 0.76 seconds. The CPU usage in percentage is dominated by OpenFace with 59%. Facenet not only leads in terms of accuracy, but with 61.05% also in terms of memory usage in percent. In both, time- and CPU measure, Facenet is in third place, making it not only the most accurate, but also computationally rather lightweight. For this reason, the prototype will use the Facenet algorithm.

Table 4.7: Computational Analysis of DeepFace Models

DeepFace Model	**Time [s]**	**CPU [%]**	**Memory [%]**
VGG-Face	1.0696 s	67.5888 %	61.7148 %
Facenet	0.7714 s	61.5466 %	61.054 %
OpenFace	0.7629 s	59.0758 %	61.433 %
Deepface	1.1557 s	62.673 %	61.6868 %
DeepID	0.6936 s	60.0223 %	61.7825 %
ArcFace	0.8293 s	63.868 %	61.7472 %

4.3.3 Testing with reduced Picture Size

The pictures that were used for the test were larger and in higher quality than what is expected from an online video conferencing tool such as Zoom or MS Teams. In order to ensure a high accuracy even under imperfect conditions, a similar test is carried out with a new set of input pictures. This time only four different faces have to be recognized, and for each reference picture, the program compares the same face in different sizes. They start at 100x100 pixels and decrease in steps of ten until 10x10 pixel as shown in figure 4.8.

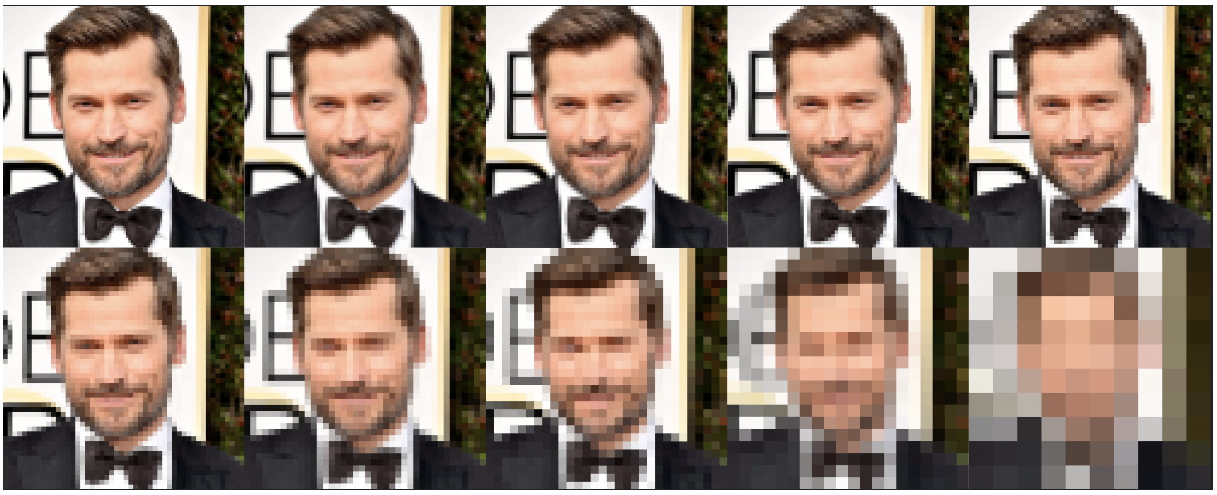


Figure 4.3: Example Collage of Input Images with reduced Size

Table 4.6 shows the accuracy of the different DeepFace models. The accuracy is measured in percent and ranges from 0%, meaning none of the four people were recognizes, to 100%.

Evaluating Picture Size	100x100	90x90	80x80	70x70	60x60	50x50	40x40	30x30	20x20	10x10
VGG-Face	100%	100%	100%	100%	100%	75%	25%	0%	0%	0%
Facenet	100%	75%	50%	50%	25%	25%	0%	0%	0%	0%
OpenFace	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
DeepFace	75%	50%	50%	50%	25%	25%	0%	0%	0%	0%
DeepID	0%	0%	25%	25%	25%	0%	0%	0%	0%	0%
ArcFace	100%	100%	75%	75%	75%	25%	0%	0%	0%	0%

Figure 4.4: True Positives in Percent per Picture Size and Method

The models VGG-Face, Facenet and ArcFace perform well in terms of true positives despite reduced picture size. VGG-Face had a total of 4 false positives, making it less reliable than the other two methods that had none. In terms of reduced image size, ArcFace performs best followed by Facenet.

4.3.4 Testing with reduced Picture Resolution

Not only the picture size can vary when using online video conferencing tools, but also the video resolution. A short drop in Wi-Fi connection, which can be expected at any time, is enough to drastically reduce the image resolution resulting in blurs and artifacts. The reduction in image resolution is called image compression and can be noticeable in extreme cases especially if a so-called lossy form of compression was used, meaning that image information is lost and cannot be restored. This is why it is important for this thesis that the face recognition also works with a low image resolution. To test this, a new set of input pictures has been prepared. Similar to the comparison with different sizes, the picture quality was gradually reduced by compressing the images to a few kilobytes.

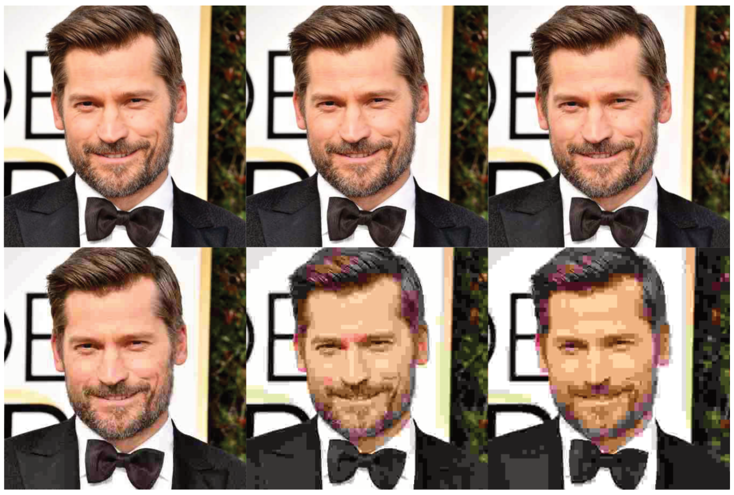
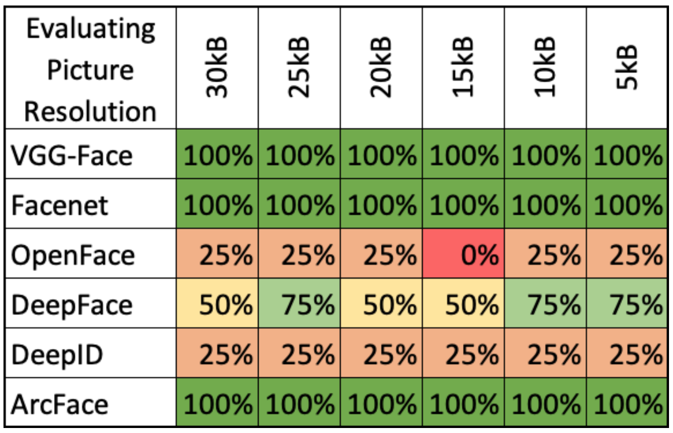


Figure 4.5: Example Collage of Input Images with Image Compression

The dataset contains only JPEG pictures for the reason that it is a lossy form of compression and thus is suited for a worst-case scenario test. Again, the same four reference pictures have to be recognized. This time however, only 6 pictures each have to be compared. The picture compression used in the test case starts at 30 kilobytes and was gradually reduces to 5 kilobytes.



Evaluating Picture Resolution	30kB	25kB	20kB	15kB	10kB	5kB
VGG-Face	100%	100%	100%	100%	100%	100%
Facenet	100%	100%	100%	100%	100%	100%
OpenFace	25%	25%	25%	0%	25%	25%
DeepFace	50%	75%	50%	50%	75%	75%
DeepID	25%	25%	25%	25%	25%	25%
ArcFace	100%	100%	100%	100%	100%	100%

Figure 4.6: True Positives in Percent per Picture Resolution and Method

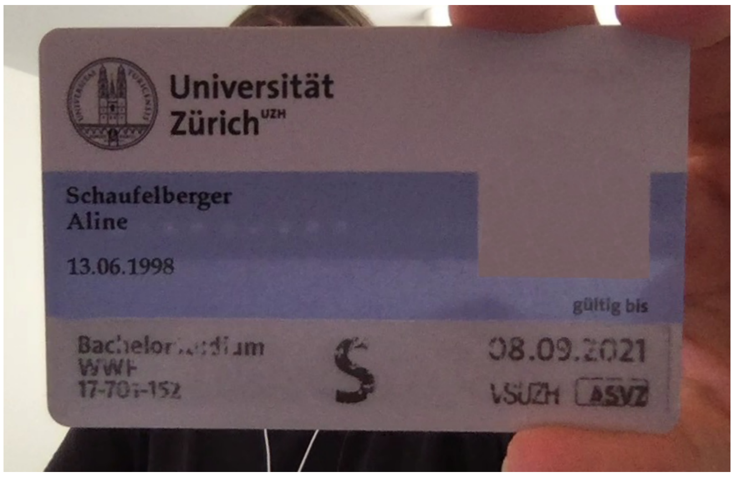
The results in the graph show that the methods perform exceptionally well. Especially VGG-Face, Facenet, and ArcFace stand out with an accuracy of 100% true positives. However, VFF-Face had a total of 5 wrong positives, making it less accurate in total. Facenet and ArcFace have no wrong positives, meaning that picture compressons down to 5 kilobytes have little to no effect on these methods.

4.4 Optical Character Recognition

A further building block towards the prototype is an optical character recognition, or short OCR. The Idea behind using OCR for this prototype was that each student will show the student ID to the camera. The OCR algorithm reads the contents of the card and will find given person in the university database. This way the student pictures from the screen capture tool will be compared to the correct student photo. This method performs a face verification.

However, this method is only applied if it has a higher accuracy than the DeepFace approach. The comparisons of the last chapter have shown that DeepFace has an accuracy of 99.26% without having to read out the name or the student number first. In addition, only one image per student is required for the face identification approach which can save time. In order to conduct the OCR test, existing methods have to be found and analyzed.

The two methods that are widely used for an accurate OCR are called PyTesseract and PyOCR. To test the methods, a dataset containing 60 pictures of UZH student ID's taken by a laptop camera is created. This allows the test to simulate students holding theirs student ID to the camera. The first trial processes the pictures directly, without any image transformations. Out of the 60 pictures, neither PyTesseract not PyOCR could recognize a single name on the student ID.



Universität
Zürich UZH

Schaufelberger
Aline
13.06.1998

gültig bis

Bachelorstudium
WWF
17-701-152

S

08.09.2021
VSUZH ASVZ

Figure 4.7: OCR input without Image Transformation

In order to achieve better results, the source pictures have to undergo several transformations as shown in the following code snippet. The image is first transformed into a numpy array. This way it can be converted into a grayscale image. Further transformations using gaussian blur and binary thesholds turn the input image into a black and white picture with high contrast and visible edges. This increases the readability of the OCR algorithm.

```
    img = np.array(Image.open(picture_folder+file))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    gaussian = cv2.ADAPTIVE_THRESH_GAUSSIAN_C
    binary = cv2.THRESH_BINARY
    img = cv2.adaptiveThreshold(img, 255, gaussian, binary, 31,
        5)
    img = cv2.medianBlur(img, 1)
    img = cv2.bilateralFilter(img, 9, 75, 75)
```

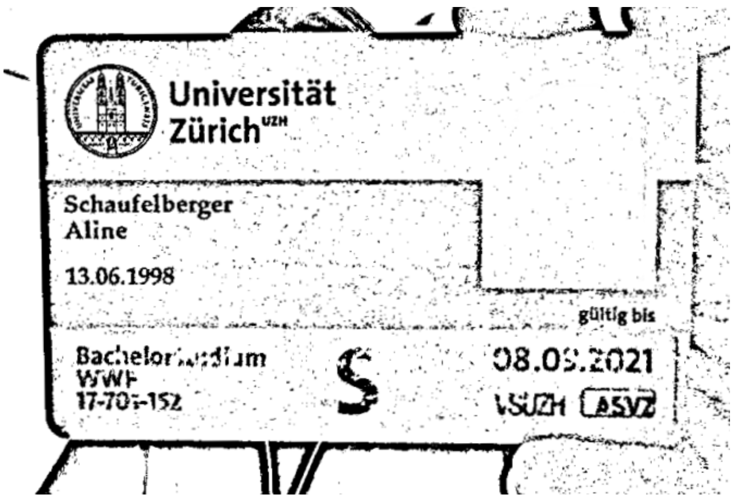

Universität Zürich UZH
Schaufelberger
Aline
13.06.1998
gültig bis
Bac:helor[illegible]
WWF
17-70[illegible]-152
08.0[illegible].2021
VSUZH ASVZ

Figure 4.8: OCR input after Image Transformation

After the transformation, PyTesseract reads the name in 20 out of 60 pictures. With 25 correct answers, PyOCR has a better performance. However, both methods have an accuracy of less than 50% and are therefore not suitable for the application. This leads to the decision to create the final prototype without OCR.

4.5 Simulated Test Environment

Once the prototype is finalized, the application has to be tested. To do so, an examination environment, shown if figure 4.9, was simulated using the online video conferencing tool "Zoom". The purpose of this trial is to evaluate the functionality and accuracy of the prototype. To preserve the identity of the participants, mockup images and names were used here.

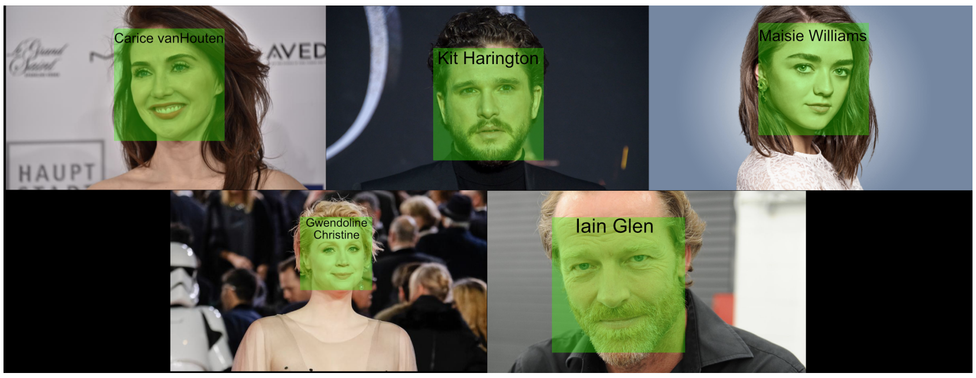
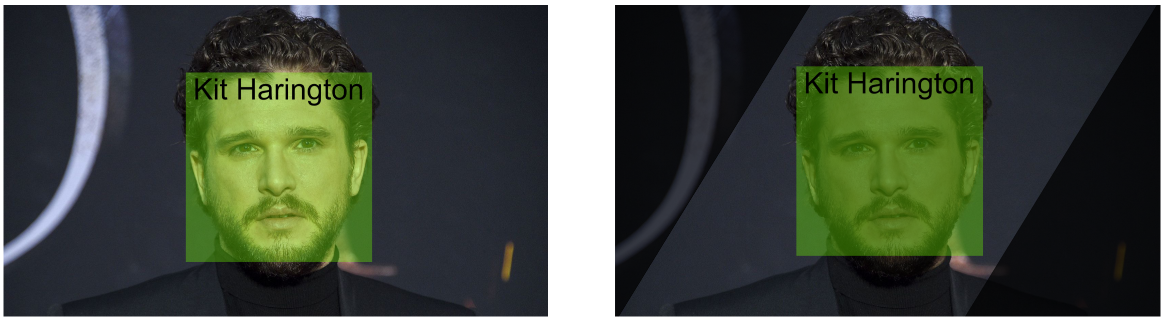


Figure 4.9: Mockup picture from the first Trial

In order to test the accuracy under different circumstances, two runs were executed. The first trial had plenty of light whereas the second one had more challenging lighting conditions. The test simulated a class of 5 people, each having one photo in the university database. The stored photos have a high resolution of 1820 x 1610 pixels with a uniform background which are good conditions for the face training. The difference in lighting conditions can be seen in figure 4.10 showing the first run (a) with good lighting conditions and the second run (b) with more challenging conditions.



(a) Well exposed Image

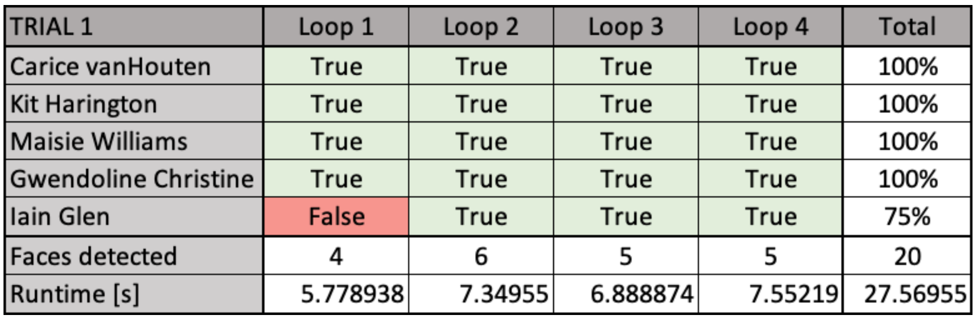
(b) Badly exposed Image

Figure 4.10: Mockup Pictures showing the Difference in Lightning Conditions between the first and second Run

Each trial consists of 4 loops. A loop is defined by the time needed to detect and recognize each face on one screenshot. Figure 4.11 and 4.12 show a more detailed examination of the experiment. Here, *True* stands for a correctly recognizing the person. *False* means that the face of the given person was not found on the screenshot. This can also be seen from the number of faces detected.

4.5.1 Trial with high Exposure

The application was able to correctly classify all faces without false positives in the first trial. Two minor mistakes that can be seen in figure 4.11 were made. The face of one student was not detected in the first loop of this trial. The other fault was detecting an additional face in the second loop. Since this was not found in the database, it was marked as ”unknown”. Detecting additional faces can occur when the backgrounds are not uniform. For the remaining cases, all faces were detected and recognized correctly which can be seen in figure 4.9.

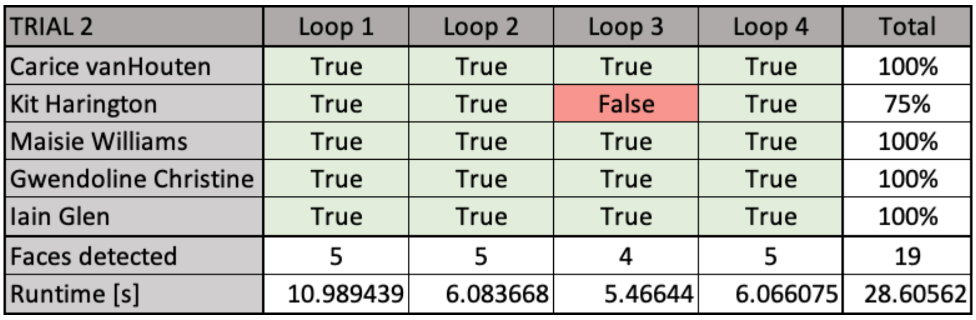


TRIAL 1	Loop 1	Loop 2	Loop 3	Loop 4	Total
Carice vanHouten	True	True	True	True	100%
Kit Harington	True	True	True	True	100%
Maisie Williams	True	True	True	True	100%
Gwendoline Christine	True	True	True	True	100%
Iain Glen	False	True	True	True	75%
Faces detected	4	6	5	5	20
Runtime [s]	5.778938	7.34955	6.888874	7.55219	27.56955

Figure 4.11: Evaluation of the first Trial

4.5.2 Trial with low Exposure

The detection rate for the second trial was expected to be lower than in the first one. Since online exams do not always have perfect lighting conditions, this second test was carried out. This should provide a more realistic recognition for online exams. However, figure 4.12 shows that the applications had no problems with the light difference and despite the poorer conditions, the application was again able to recognize all 5 people with only one minor mistake. During the third loop, a face was not detected and could therefore not be found in the database.



TRIAL 2	Loop 1	Loop 2	Loop 3	Loop 4	Total
Carice vanHouten	True	True	True	True	100%
Kit Harington	True	True	False	True	75%
Maisie Williams	True	True	True	True	100%
Gwendoline Christine	True	True	True	True	100%
Iain Glen	True	True	True	True	100%
Faces detected	5	5	4	5	19
Runtime [s]	10.989439	6.083668	5.46644	6.066075	28.60562

Figure 4.12: Evaluation of the second Trial

This test showed that the prototype provides a solution that fulfills the requirements for identity verification in an online environment. The persons could mostly be identified correctly in good lighting conditions as well as in low lighting. However, if this application is to be used for large classes of several hundred of students, it is recommended to test it with a larger number of individuals first.

Chapter 5

Summary, Conclusion, and Future Work

After the detailed reporting on the carried out work regarding analysis, implementation and evaluation of the subject matter, it is the moment to summarize, conclude and to consider future work. The conclusion shall answer the question that this posed in this thesis.

5.1 Summary

Can an online identity verification based on face recognition be used in online exams? This thesis not only answered this question but also developed a prototype to prove the chosen concept.

Requirements of this application are runing on one machine only and being independent from video communication tools. Moreover, a user expects to detect and recognize multiple faces simultaneously. Additionally, the application shall give an intuitive real-time feedback. The chosen design and methods should provide a high accuracy for the identity verification, a short run time and low computational cost.

Reading directly from screen memory made the application independent from online video conferencing tools and moreover allowed to verify the identities of multiple individuals simultaneously. The breakthrough of this thesis consisted in a change of approach. Instead of taking the screen input as reference pictures, an improved face recognition algorithm allowed to use only a single image as a reference. This led to the possibility of using the student photos in the university database for the face encodings. Thus, a real-time face recognition could be created, reducing the need for human intervention. Another advantage of this approach includes increased data security. If the learning of the reference pictures can be done in advance, it could already be done by university computers. Thus, an examiner would not need direct access to student pictures but only their face encodings.

Furthermore, it turned out that the verification process can be done without using optical character recognition to read content on student ID cards. The identity of an individual can be determined directly and with a higher accuracy using the new face recognition approach.

The prototype has been successfully conceptualized, developed, and tested. The simulated examination environment has shown that it is in compliance with the requirement specifications that were defined at the beginning.

5.2 Conclusion

The validation by a simulation exam proved that it is feasible to use face recognition to identify individuals in an online environment. With this, the fulfillment of the user and design requirements were reviewed and the usability of the software application in real life confirmed.

5.3 Future Work

Since a prototype for an application was implemented in this thesis, it is implied that it can be further developed and finalized in the future, such that it can be used by lecturers and exam administrators at the UZH or other universities. This application would require an interface to the university database. In addition, an intuitive front end could be helpful. This should include a simple and elegant way to switch classes if the identity of multiple classes needs to be verified.

With a little more computing power, this application could detect more than just the identity of individuals. There exist modules, that can be used to assess a person's age, gender, ethnicity, and even emotional state. With such a real-time face analysis, one could detect whether a person is happy, sad, scared, surprised, or neutral. This could give the examiner an insight into the emotional state of the students. Whether this is helpful or triggers an increase in stress level in the students could be investigated and clarified in another thesis.

Bibliography

[1] Al Sweigart. *PyAutoGUI*. Version 0.9.53.

[2] A.W. (Tony) Bates. *Teaching in a Digital Age*. 2019.

[3] Peter N. Belhumeur, Joao P. Hespanha, and David J. Kriegman. "Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection." In: *ECCV (1)*. Ed. by Bernard F. Buxton and Roberto Cipolla. Vol. 1064. Lecture Notes in Computer Science. 1996, pp. 45–58.

[4] BioID. *Face Recognition Liveness Detection*. 2021. URL: `https://www.bioid.com/` (visited on Aug. 24, 2021).

[5] Georgescu Daniel. "A real-time face recognition system using eigenfaces". In: *Journal of Mobile* (2011), pp. 193–204.

[6] Docupile. 2021. URL: `https://www.docupile.com/optical-character-recognition-ocr/#` (visited on Aug. 24, 2021).

[7] Doxygen. *OpenCV Documentation*. 2020. URL: `https://docs.opencv.org/4.5.1/db/d7c/group__face.html` (visited on Aug. 24, 2021).

[8] ExamSoft. 2021. URL: `https://examsoft.com/` (visited on Aug. 24, 2021).

[9] University of Exeter. *Facial Recognition Technology Market Research*. Tech. rep. European Regional Development Fund, 2019.

[10] Anna Katharina Fitze. *AVVS - Automated Video UZH-ID Verification System*. 2020.

[11] Gary B. Huang et al. *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments*. Tech. rep. 2012.

[12] Ian Jolliffe. "Principal Component Analysis". In: *International Encyclopedia of Statistical Science*. Ed. by Miodrag Lovric. 2011, pp. 1094–1096.

[13] Sasan Karamizadeh, Shahidan Abdullah, and Mazdak Zamani. "An Overview of Holistic Face Recognition". In: *International Journal of Research in Computer and Communication Technology* 2 (Sept. 2013), pp. 738–741.

[14] Trupti M. Kodinariya. *Hybrid Approach to Face Recognition System using Principle component and Independent component with score based fusion process*. 2014.

[15] Jan Bergstra Karl de Leeuw. *The History of Information Security - A Comprehensive Handbook*. 2007, p. 265.

[16] Gurpreet Singh Lehal and C. Singh. "Feature Extraction and Classification for OCR of Gurmukhi Script". In: 2006.

[17] Mayra Oliveira, Antonio Penedo, and VinÃcius Pereira. "Distance education: advantages and disadvantages of the point of view of education and society". In: *Dialogia* (Aug. 2018), pp. 139–152.

[18] Divyarajsinh N. Parmar and Brijesh B. Mehta. "Face Recognition Methods & Applications". In: *Computer Technology & Applications* 4.1 (2013), pp. 84–86.

[19] Aparna Patil. "Optical Character Recognition Implementation using Pattern Matching". In: *International Journal for Research in Applied Science and Engineering Technology* 7 (Aug. 2019), pp. 1092–1095.

[20] PeopleCert. 2021. URL: `https://www.peoplecert.org/exams-peoplecert-online-proctoring-windows` (visited on Aug. 24, 2021).

[21] NIRPY Research. *Classification of NIR spectra by Linear Discriminant Analysis in Python.* 2018. URL: `https://nirpyresearch.com/classification-nir-spectra-linear-discriminant-analysis-python/` (visited on Aug. 24, 2021).

[22] Giampaolo Rodola. *psutil documentation.* 2009-2021. URL: `https://psutil.readthedocs.io/en/latest/` (visited on July 31, 2021).

[23] R. Smith. "An Overview of the Tesseract OCR Engine". In: *Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02.* ICDAR '07. 2007, 629â633.

[24] SMOWL. *Enhance the quality assurance of your online assessments.* 2021. URL: `https://smowl.net/en/` (visited on Aug. 24, 2021).

[25] Statista.com. *Facial Recognition Market Size Worldwide in 2020 and 2025.* 2020. URL: `https://www.statista.com/statistics/1153970/worldwide-facial-recognition-revenue/` (visited on July 5, 2021).

[26] Yaniv Taigman et al. "DeepFace: Closing the Gap to Human-Level Performance in Face Verification". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2014, pp. 1701–1708.

[27] Alaa Tharwat et al. "Linear discriminant analysis: A detailed tutorial". In: *AI Communications* 1.22 (2017), p. 22.

[28] WISEflow. *WISEflow - More than paperless.* 2021. URL: `https://europe.wiseflow.net/` (visited on Aug. 24, 2021).

[29] G. Md. Zafaruddin and H. S. Fadewar. "Face Recognition Using Eigenfaces". In: *Computing, Communication and Signal Processing.* 2019, pp. 855–864.

Abbreviations

ASCII	American Standard Code for Information Interchange
BPNN	Back-Propagation Neural Networks
CPU	Central Processing Unit
DRS	Design Requirement Specification
FLD	Fisher's Linear Discriminant
GUI	Graphical User Interface
LDA	Linear Discriminant Analysis
LFW	Labeled Faces in the Wild
MSS	Multiple Screen Shots
OCR	Optical Character Recognition
OpenCV	Open Source Computer Vision
PCA	Principal Componant Analysis
Psutil	Python System and Process Utilities
URS	User Requirement Specification

List of Figures

List of Tables

Appendix A

Contents of the ZIP-File

The enclosed ZIP-file contains the following content:

- **Code**

  This folder contains the following python files.

  1. **UZH-Verifier.py**
     Final Prototype

  2. **UZH-Verifier_Variables.py**
     This document contain variables that are used for the final prototype

  3. **Frame_compare.py**
     Testing file for the screen capture method comparison

  4. **Recognition.py**
     Testing file for the face recognition comparison

  5. **Recognition_Variables.py**
     This document contains variables that can be changed for the test file

  6. **OCR_Compare.py**
     Testing file for the OCR comparison

- **Documentation**

  A short documentation is given in the form of a text file.

- **Literature**

  This folder contains all referenced papers.