An Automated Attendance Register

Irvin Wesso

Thesis presented in fulfilment
of the requirements for the degree of
Bachelor of Science Honours
at the University of the Western Cape

Supervisor: Mehrdad Ghaziasgar
Co-supervisor: Reg Dodds

This version November 18, 2018

Declaration

I, IRVIN WESSO, declare that this thesis "*An Automated Attendance Register*" is my own work, that it has not been submitted before for any degree or assessment at any other university, and that all the sources I have used or quoted have been indicated and acknowledged by means of complete references.

Signature: .... Date: ....

IRVIN WESSO.

Abstract

An attendance register to record students' presence or participation in a lecture, tutorial or practical is usually done manually. This traditional attendance record system, where students have fill in their details on paper, is inefficient and students can cheat by getting their peers to sign on their behalf when they are absent for a lecture. This paper introduces a practical system for recording attendance automatically using facial recognition. The first part of the system is facial detection, which is achieved by using the Viola-Jones algorithm. The second part of the system, face recognition, is achieved through feature extraction and classification trained on the cropped Yale face database. A model is built by splitting 2452 samples from 42 people by a ratio of 3:1 for training and testing sets. Eigen faces are extracted using principal component analysis and are then fed into a support vector machine for training. The third part of the system uses the model to predict the student who enters the class and then records his/her attendance in a spreadsheet.

Key words

Class attendance

Eigen faces

Face recognition

Principal components analysis

Real-time attendance register

Viola-Jones algorithm

Support Vector Machine

Haar cascade

Register

F1-Score

Acknowledgment

This thesis is a compilation of the efforts of many people that helped me through the years. I would first like to thank my supervisor Dr Mehrdad Ghaziasgar and Reg Dodds for encouraging me during my study. Without our weekly meetings, this work would not have been possible. I would like to extend a very special thanks to NRF for funding my BSc(Hons) Computer Science year. . . .

Contents

List of Tables

List of Figures

Glossary

Principal Components Analysis (PCA) . Statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.

Support Vector Machine (SVM) . Supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis.

OpenCv . Computer Vision Library

RGB . Red Green Blue

Radial Basis Function (RBF) . Real-valued function whose value depends only on the distance from the origin,

FP . False Positive

TP . True Positive

Chapter 1

Background

An attendance register is an official list of people who are enrolled in a course and who are expected to be present at an institution such as a school, university or college. An attendance register is a tool used to record students' presence or participation in a lecture, tutorial or practical. The register contains a list of students' names and their student number. In each lecture, the register circulates in the class and students are required to sign next to their name to mark their presence. Once all students present have signed next to their name, the register is returned to the lecturer who then inputs the attendance into a database. There is evidence that there is a significant correlation between students' attendances and their academic performance (Newman-Ford et al., 2008). Othman et al., (2009), claim that those students who have poor attendance records tend to present poor retention.

Computer Science lecture CSC312			
Date: 14/04/2018	Student Number	Name & Surname	Signature

Figure 1.1: Example of attendance register

1.1 Problem Statement

Students' attendance is recorded by most universities and and is required by law in state schools, and each faculty has to maintain proper records for attendance. The traditional attendance record system, where students have to manually fill in their details in Figure 1.1, is inefficient and requires more time to do analysis on a student's attendance. Many students are helping their peers by signing their attendance when they are absent for a lecture. In many cases students come to class late which results in the register not being circulated throughout the entire class. If the attendance register gets lost or destroyed, the lecturer has to print a new attendance register and recirculate it. This is not ideal because previously absent students have a chance to mark their presence on the new register. Time is expended on completing manual attendance registers and students could miss key aspects of a lecture while filling in their details. Manual attendance registers for large classes do not work because once the register in Figure 1.1 is full, students have to draw extra lines, which is often untidy, to create space to fill in their details.

1.2 Proposed Solution

All the problems mentioned above can be solved by creating an automated attendance register system that uses facial recognition. A camera will be mounted at the entrance of a classroom so that when a student enters, his/her image is captured by the camera. The Viola-Jones algorithm is applied to detect the face which is then resized and enhanced by using linear stretch contrast enhancement. Finally, a machine learning technique called PCA/LDA is used to recognize the faces. Once the face is recognized, attendance will automatically be updated in a spreadsheet along with his/her name, date and time. The main goal is to create a system that is practical, reliable and eliminates fraudulent signatures, disturbance and time loss in traditional attendance systems. A further goal is to do analysis on a student's attendance and the impact it has on their performance.

1.3 Assumptions

To ensure optimal results, the following assumptions have been made:

- Lectures will be during the day, in a well-lit classroom.
- The camera will be placed at the entrance of a classroom.
- Students will not get haircuts or change how their face looks.
- Students will look directly into the camera so that it can capture their face.
- There will only be one entrance to the classroom.

Chapter 2

Related work

2.0.1 Class Room Attendance system using facial recognition system—Abhishek Jha

Providing an automated attendance register system that resolves issues mentioned in the problem statement. The idea is to have a camera at the entrance of a class recording a video of all students that attend a certain lecture, laboratory or exam and compiling an attendance register. The main objective of this system is to provide a system that is practical, reliable and eliminates disturbance and time loss in traditional attendance systems. A further objective is to present a system that can evaluate students' performances depending on their attendance rate.

Implementation and Technologies used:

Techniques such as color-based detection and Principle Component Analysis (PCA) for face detection and for feature extraction, PCA and Linear Discriminate Analysis (LDA). For detection, colour based technique was implemented, which depends on the detection of the human skin color with all its different variations in the image. The skin area of the image is then segmented and passed to the recognition process. For recognition, PCA technique has been implemented which is a statistical approach that deals with pure mathematical matrices not image processing like the colour-based technique used for detection. PCA can also be used for detection (Jha, 2007).

2.0.2 Automated Attendance Management System Based On Face Recognition Algorithms—Chintalapati and Raghunadh

When a person enters the classroom his/her image is captured by a camera at the entrance. The face region is then extracted and pre-processed for further processing. When the student's face is recognized it is fed to post-processing.

Implementation and Technologies used:

A) *Image Capture*—the camera is mounted at the entrance of the lecture hall in order to capture a frontal image of a student entering the room. The image size is preferred to be the size of 640×480 to avoid back-end resizing which can result in poor performance. (Chintalapati and Raghunadh, 2013)

B) *Face Detection*—the Viola-Jones algorithm was used as it has a high decision rate and is fast and robust. It makes use of the *integral image* and *AdaBoost* learning algorithm as classifier. Chintalapati and Raghunadh observed that this algorithm gives better results in different lighting conditions and angles.

C) *Pre-Processing*—histogram equalization is used on the extracted face image and is resized to 100×100. This method improves the contrast of the images, making it clearer.

D) *Database Development*—images of individuals were taken at different angles, expressions and lighting conditions. A database of 80 individuals with 20 images of each was collected for the project which was stored in the database.

E) *Feature Extraction and Classification*—principal component analysis (PCA) was used to represent facial images using eigen faces. The formula $X = WY + \mu$ was used to represent an image mathematically. X is the vector, Y is the vector of eigen faces, W is the feature vector and μ is the average face vector.

F) *Post Processing*—after the students' faces are recognized, their names are updated in an excel spreadsheet.

2.0.3 Development of a Student Attendance Management System Using RFID and Face Recognition: A Review—Patel and Priya

A CCTV camera which is fixed to the entry point of a classroom captures the image of a person and checks the observed image with the face database using an android enhanced smartphone. It is typically used for marking attendance for students and people who are strange to the environment, i.e., unauthorized persons.

Implementation and technologies used: The model is developed with the help of OpenCV library. Viola-Jones algorithm is algorithm is used for detecting human faces which is then resized to the required size. The resized face is then further processed using linear contrast enhancement. PCA/LDA is used to recognized the image. When the recognition is done, attendance will be automatically updated on a spreadsheet with his/her name, time and date. An HTML file is then automatically updated by their system so that a remote authenticated user can access the attendance file (Patel and Priya, 2014).

2.0.4 Real Time Face Recognition Using AdaBoost Improved Fast PCA Algorithm–Kumar et al.

Detect real time human faces using AdaBoost with Haar cascade and a simple fast PCA and LDA is used to recognize the faces detected. The matched face is then used to mark attendance in the laboratory. This biometric system is a real-time attendance system based on human face recognition with a simple and fast algorithm and gaining a high accuracy rate (Kumar et al., 2011).

Implementation and Technologies used: An input image is taken through a web camera continuously till the system is shutdown. The image is then cropped by a face detection module which saves the facial information in JPEG format of 100×100 coloured matrix size with three layers, i.e., *red*, *green* and *blue*.

Face Detection:

A `opencv1.sln`: This is a solution file which calls all other files. This `.sln` is created whenever we create a web application or any application in MS Visual Studio.net. This file provides the editing facility in the code.

B `prog1.cpp`: It is the main program file in the face detector module. It detects the face and crops the face image and saves in the current folder in which it is.

C `haarcascadefrontalfacealttree.xml`: It is a cascade file in XML used to obtain Haar cascade for the frontal face in the image. It is used in the OpenCV library.

D `StudentAttendence.xls`: It records the attendance of the detected face according to the system time in excel sheet.

E `StudentAttendence.doc`: It is same as the above file; the only difference is that it saves the records in document format which can be easily printed for the detailed information.

Face Recognition:

A `example.m`: It is the first page to be shown to the user. It calls the other files in this module. It takes input training dataset and also inputs the test dataset.

B `CreateDatabase.m`: This module is in Matlab used to create database for the face images in the training dataset in a sequence of increasing numbers as the face images in the dataset are in number format.

C `EigenfaceCore.m`: This module in the face recognition stage calculates the eigen face value using PCA and then applying the LDA algorithm on the result of PCA.

D `facerec.m`: This creates graphical interface in Matlab for training and testing the database.

E `Recognition.m`: This function compares two faces by projecting the images into face space and % measuring the Euclidean distance between them.

F `facerec.exe`: This is the executable file created to linked the Matlab files with MS VS .NET 2008. It works in same way as the Matlab files does.

Chapter 3

Image Processing Techniques

3.1 Introduction

This chapter looks at the various image processing techniques used to extract features of the face to create a classifier. OpenCV is a computer vision and machine learning library which uses the Viola-Jones algorithm to detect the location of faces in an image. Using the location, the face is extracted and gray scaled in order to extract features of the face. The gray-scaled image is then fed into a support vector machine to train and to create a classifier to recognize the face of a student.

3.2 Viola-Jones Algorithm

3.2.1 Integral Image

The first step is to take an input image and convert it into an integral image. This can be achieved by making each pixel in the image equal to the entire sum of all the pixels above and to the left of the concerned pixel (Jensen, 2008). Figure 3.1 demonstrates this. The integral image

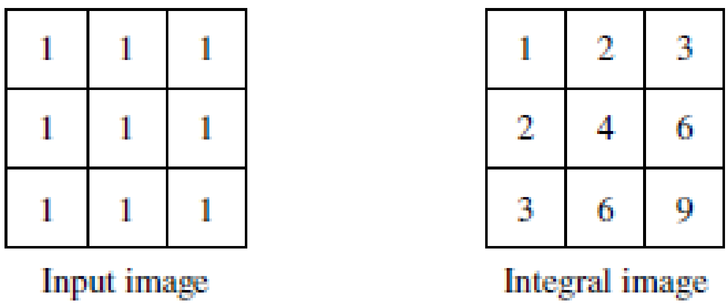

Figure 3.1: Integral image

allows for the calculation of the sum of all pixels inside the given rectangle using only four values. These values are the pixels in the integral image that coincide with the corners of the rectangle in the input image. Figure 3.2 demonstrates this. Rectangles B and C include the rectangle

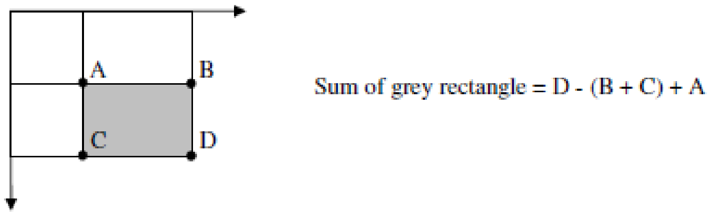

Figure 3.2: Sum calculation

A, therefore the sum of A should be added to the calculation as shown in Figure 3.2.

3.2.2 Haar Features

Since the sum of pixels within rectangles of arbitrary size can be calculated in constant time, a given sub window can be analysed using features consisting of two or more rectangles (Jensen, 2008). These features are called Haar features and there are different types as shown in Figure 3.3.

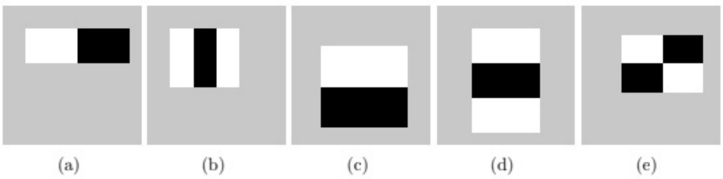

Figure 3.3: Five Haar-like patterns

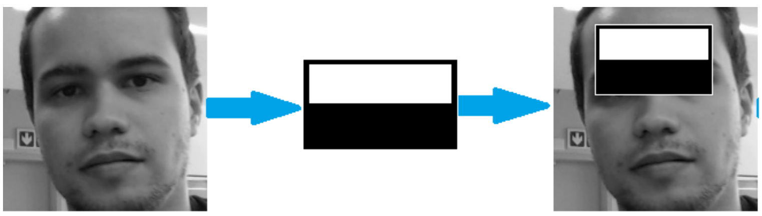

Figure 3.4: Haar-like features of the face

The size and position of a pattern's support can vary provided its black and white rectangles have the same dimension, border each other and keep their relative positions. Thanks to this constraint, the number of features one can draw from an image is manageable: a 24×24 image, for instance, has 43200, 27600, 43200, 27600 and 20736 features of category (a), (b), (c), (d) and (e) in Figure 3.3 respectively, hence 162336 features in all. These features hold the information to characterize the face (Wang, 2014).

3.2.3 AdaBoost Algorithm

Among the 162336 features as stated above, many are expected to give almost consistently high values when positioned over a face. In order to find these features Viola-Jones uses a modified version of the AdaBoost algorithm developed by Freund and Schapire in 1996 (Wang, 2014).

AdaBoost is a machine learning boosting algorithm that constructs a strong classifier through a weighted combination of weak classifiers. A weak classifier classifies correctly in slightly more than half the cases. The AdaBoost algorithm reduces a large feature set down to a smaller set of important features. A mathematical representation of a weak classifier is one where

$$h(\boldsymbol{x}, f, p, \theta) = \begin{cases} 1 & \text{if } pf(\boldsymbol{x}) > p\theta, \\ 0 & \text{otherwise}, \end{cases}$$

where $\boldsymbol{x}$ is a 24×24 pixel sub-window, f is the applied feature, p the polarity and θ the threshold that decides whether x should be classified as a face or a non-face (Jensen, 2008).

3.2.4 Cascaded classifier

The cascaded classifier consists of many stages, each containing a strong classifier. These stages determine whether a given sub-window is definitely not a face or *maybe* a face. When a sub-window is classified to be a non-face by a given stage, it is discarded. If it is classified as a *maybe*-face it is passed on to the next stage in the cascade. The more stages a given sub-window passes, the higher the chance the sub-window

Given a set of n training examples

$$\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\},$$

where each $\boldsymbol{x}_i$ with $i \in [1..n]$ is an input vector of the training features and labelled with $y_i \in \{0, 1\}$. There are l positive and m negative examples.

(a) Initialize the weights of every sample, $w_{1,i} = \begin{cases} \frac{1}{2l} & \text{if } y_i = 0, \\ \frac{1}{2m} & \text{otherwise.} \end{cases}$

(b) For each weak classifier $t = 1, \ldots, T$

i. Normalize the weights to probabilities, $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,i}}$.

ii. Select the best weak classifier with respect to the weighted error

$$\varepsilon_t = \min_{f,p,\theta} \sum_i w_{t,i} |h(\boldsymbol{x}_i, f, p, \theta) - y_i|.$$

iii. Define $h_t(x) = h(x, f_t, p_t, \theta_t)$, where F_t, p_t, and θ_t are the minimizers of ε_i.

iv. Update the weights $w_{t+1,i} = w_{t,i}\beta_t^{1-\varepsilon}$, where $\beta_t = \frac{\varepsilon_i}{1-\varepsilon_i}$ and $e_i = 0$ if example $\boldsymbol{x}_i$ is classified correctly, and $e_i = 1$ otherwise.

v. The final strong classifier is

$$C(x) \quad = \quad \begin{cases} 1 & \text{if } \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t, \\ 0 & \text{otherwise,} \end{cases}$$

where $\alpha_t = \log \frac{1}{\beta_t}$.

Figure 3.5: Pseudo code for the AdaBoost algorithm

actually contains a face. Below is an illustration of the concept (Bishop, 2016).

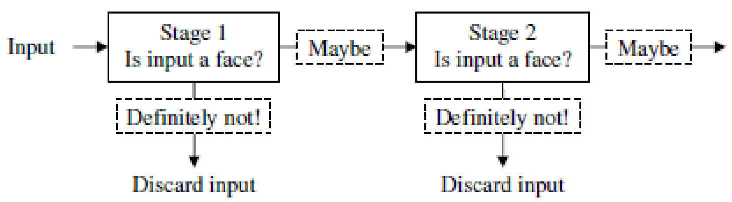

Figure 3.6: Cascade Classifier

3.3 Pre-Processing

3.3.1 *RGB* to Grayscale

To make image processing easier, the *RGB* colour output from the camera is converted to a grayscale image. This reduces the number of colour channels to a single channel—gray scale is represented using one-pixel value compared to RGBs three. RGB is converted to chrominance, its colour and luminance, its intensity at each pixel. Grayscale is represented using each pixel's luminance value and is calculated as the sum of each RGB colour multiplied by a weight and is calculated by

$$Y = 0.2126R + 0.7152G + 0.0722B$$

where R, G and B are values from 0 to 255 (Jensen, 2008).

3.3.2 Re-sizing

As discussed in the previous section, re-sizing the image scales down the number of pixels in an image which results in a smaller feature set. Re-sizing the image also ensures uniformity in our feature set (Bishop, 2016).

Chapter 4

Implementation

4.1 Introduction

This chapter looks at high-level and low-level implementation of the automated attendance register system. The high-level view in Section 4.2 provides a brief outline of the processes followed during the implementation of the system, while the low-level view in Section 4.3 goes into more detail about the implementation of the system.

4.2 High Level Implementation

This section gives an overview of the various stages that the system follows. The system consists of seven stages, namely, *capture frames*, *detect face*, *capture face*, *extract features*, *train machine learning algorithm*, *recognize face* and *record attendance*. A visual representation of these stages can be seen in Figure 4.1.

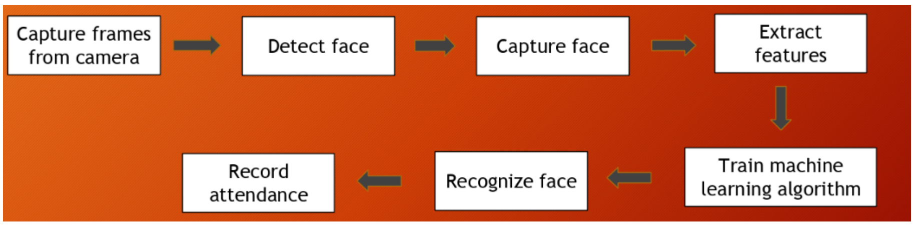

Figure 4.1: High Level Implementation

- **Capture Frames**—A Logitec camera captures a continuous stream of video input which is displayed on the screen.
- **Detect Face**—A single frame is captured in order to check if there is a face present by using a face detection algorithm. Once a face is detected, a rectangle is drawn around it.
- **Capture Face**—Once the face is detected, it needs to be captured and stored in order to extract features for training in the next stage. The lecturer will take 15–25 images of each student in the classroom.
- **Extract Features**—Each face captured has a unique set of features that can be used to identify a student. Unique features are extracted using eigen faces so that the computer can understand and process these features. The feature extraction is applied to the stored facial image captured in the previous step (Turk and Pentland, 1991).
- **Training**—Machine learning builds algorithms that can receive input data and use statistical analysis to predict an output while updating outputs as new data becomes available. By training the system using the extracted features, it will have the ability to make predictions that is explained in the next stage (Domingos, 2012).
- **Recognize Face**—A model is built based on the trained features that is used to predict and recognize a face (Domingos, 2012).
- **Record Attendance**—Once the face is recognized, attendance is recorded in a spreadsheet.

4.3 Low-level Implementation

Figures 4.2 and 4.3 contain a detailed overview of the low-level implementation of the system. The various stages are explained in this section.

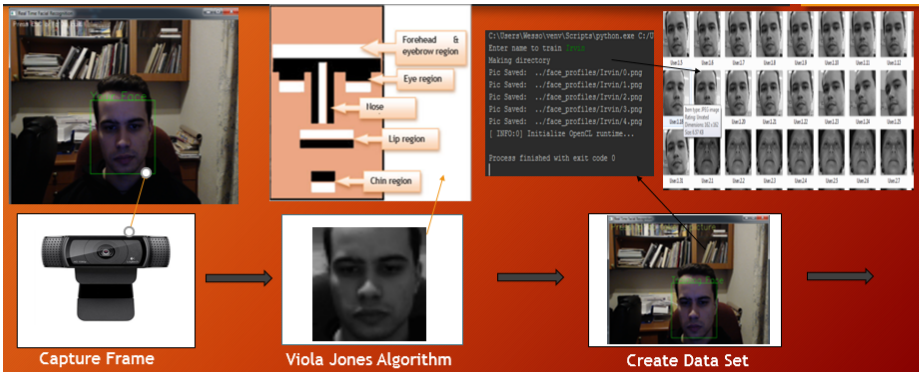

Figure 4.2: Low-level implementation

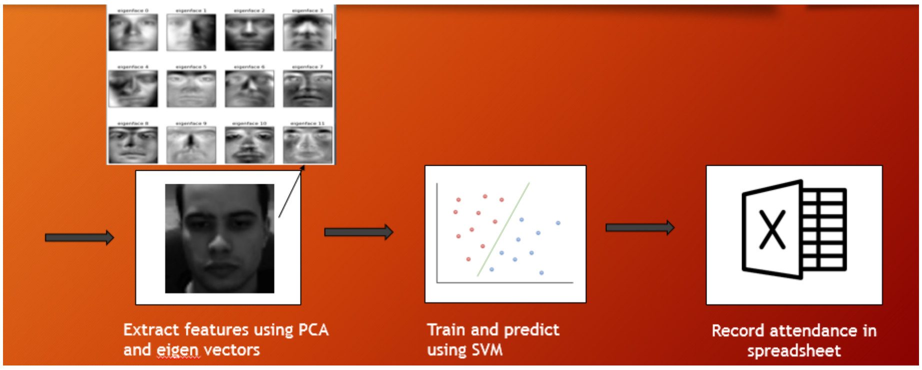

Figure 4.3: Low-level implementation continued

4.3.1 Capture Frame

`OpenCv`, which is an open source computer vision library, is used to capture a continuous stream of images from the webcam and a live video feed will appear on the screen as shown in Stage 1 of Figure 4.2.

4.3.2 Detect Face

As explained in Section 3.2, the Viola-Jones algorithm is applied to detect the face and a rectangular box is drawn around the facial region as shown in Figure 4.2.

4.3.3 Capture Face

Once the face is detected, the user enters the student's name, surname, student number and captures 15–25 images by pressing "p" on the keyboard. A separate folder is created to store all the images of each individual student as shown in the third stage of Figure 4.2.

4.3.4 Extract Features

Eigen Features using eigen vectors and PCA will be extracted from the folders created where the images are stored. These features are extracted so that the computer can understand and process them. An example of these faces can be seen in Figure 4.3.

4.3.4.1 Eigen Faces

Eigen faces is an appearance-based approach to face recognition that seeks to capture the variation in a collection of face images and use this information to encode and compare images of individual faces in a holistic. Eigen faces are the principal components of a distribution of faces, or equivalently, the eigen vectors of the covariance matrix of the set of face images, where an image with N pixels is considered a point (or vector) in N-dimensional space. The figure below is an example of how PCA is used to extract the eigen vectors of a face (Turk and Pentland, 1991).

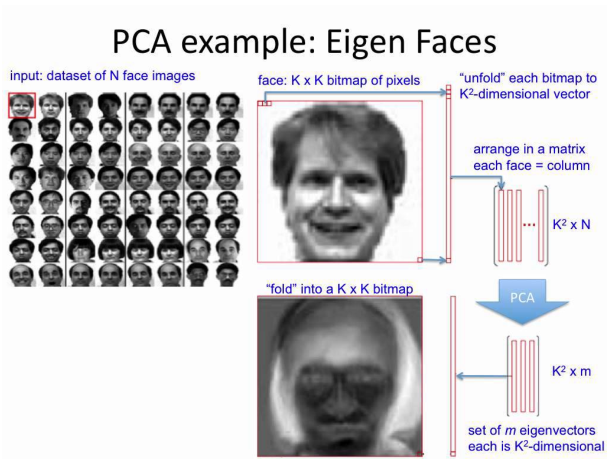

Figure 4.4: PCA example to extract eigen vectors

4.3.5 Train Machine Learning Algorithm

The eigen faces may be considered as a set of features which characterize the global variation among face images. Then each face image is approximated using a subset of the eigen faces, those associated with the largest eigenvalues. These features account for the most variance in the training set (Domingos, 2012).

4.3.5.1 Support Vector Machine

A *support vector machine* (SVM) is a supervised machine learning algorithm that can be employed for both classification and regression purposes. This technique was used in this system because it has been proven to be accurate with face and facial action detection. The kernel used for training is the *radial basis function* (RBF). Support vectors are those data points nearest to the separating hyperplane, the points of a data set that, if removed, would alter the position of the dividing hyperplane. Because of this, they can be considered the critical elements of a data set. A hyperplane is a surface that linearly separates and classifies a set of data as seen in Figure 4.5. When new testing data is added, whatever side of the hyperplane it lands will decide the class that we assign to it. It is important to choose a hyperplane with the greatest possible margin between the hyperplane and any point within the training set, giving a greater chance for new data to be classified correctly as seen in Figure 4.6 (Noble, 2006).

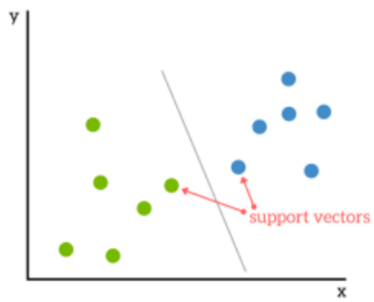

Figure 4.5: Data plane dividing two data sets

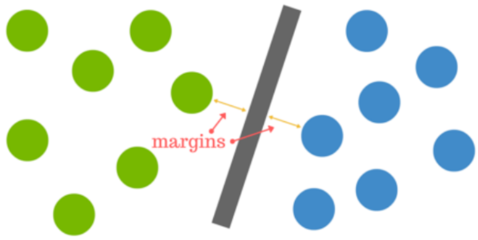

Figure 4.6: Greatest possible margin

4.3.6 Record Attendance

When a student walks into a class and their face is recognized, the name, surname, student number, date and time of entry is recorded in a CSV file which can be opened in a spreadsheet or text editor.

Chapter 5

Testing and Optimization

5.1 Introduction

This chapter covers the optimization of the system to obtain greater accuracy and efficiency. This chapter also discusses the results from the training and testing the system.

5.2 Optimization

5.2.1 Student Verification

Attendance is only recorded when the system recognizes the same student for five continuous frames. This helps the system to avoid incorrectly recognizing and recording students attendance. For example, if a student is recognized as “John” for two frames and in the third frame he is incorrectly recognized as “Smith” the system will not record his attendance. If the student is recognized as “John” for five continuous frames, the system confirms it has correctly predicted the student and attendance is recorded.

5.2.2 Handling Unknown Faces

If an unknown student enters a classroom the system would incorrectly predict the student as one of the students in the data set. This is because a SVM always outputs a student with the highest prediction accuracy. After running a few tests, it was found that the “forced” prediction accuracy of an unknown student was low and a threshold had to be determined so that when the prediction of a student is below a certain accuracy the system records the student as unknown. How this threshold was determined is discussed in Sections 5.2.2.1 and 5.2.2.2.

5.2.2.1 Using probabilities of known and unknown faces to determine a threshold

A subset of 469 samples of unknown faces from the Yale data set were fed into the system and the probabilities of their "forced" prediction were documented and graphed in Figure 5.1. It was found that majority of the forced probabilities of the unknown faces were *below 50%* as shown in Figure 5.1. This result indicates what the value of the threshold should be.

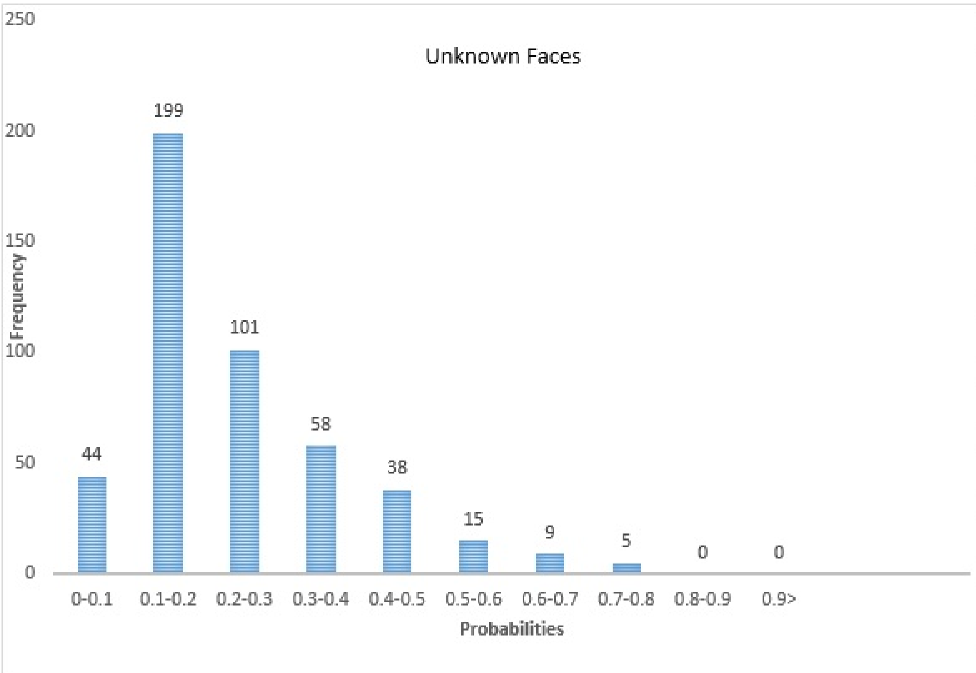

Figure 5.1: Forced probabilities of unknown faces

A subset of 2471 samples of known faces from the Yale data set were fed into the system and their prediction probabilities were graphed in Figure 5.2. It was found that majority of the probabilities were *above 50%* as shown in Figure 5.2. These results, along with the results in Figure 5.1 confirmed that the *rejection threshold* should be between 0.4 and 0.5.

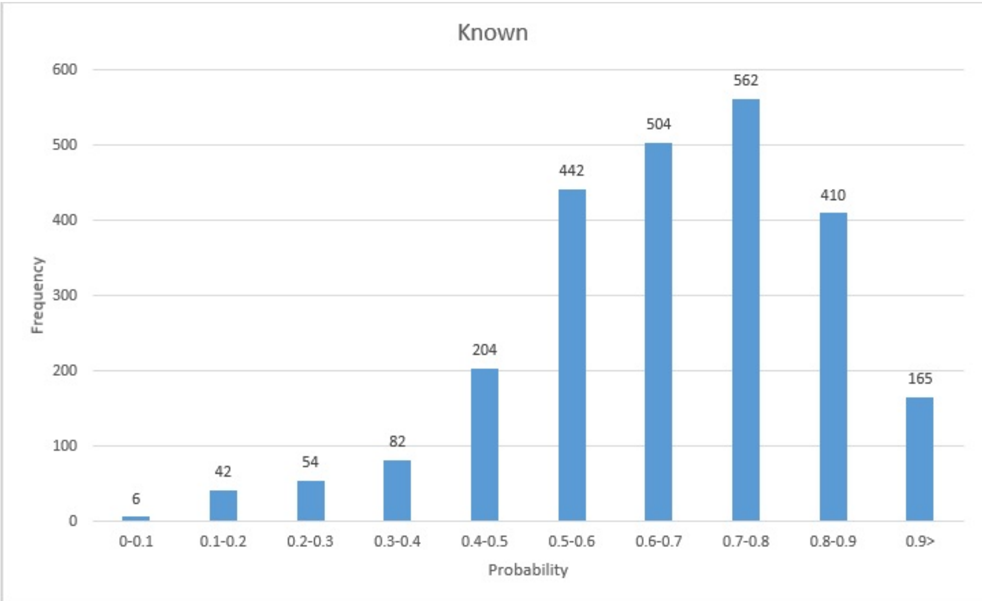

Figure 5.2: Probabilities of known faces

5.2.2.2 Calculating the F1 score to determine a threshold

The F1 score is the harmonic average of the precision and recall. The F1 score reaches its best value at 1, which indicates perfect precision and recall and worst at 0. Table 5.1 shows formulae to calculate and evaluate the F1 score of a SVM model.

Table 5.1: Calculating the F1-Score

Evaluating SVM model		
Term	Formula	Description
Type 1 Error	FP	False Positive
Type 2 Error	TP	True Positive
Recall	$Recall = \frac{TP}{TP+FN}$	True Positive Rate
Precision	$Precision = \frac{TP}{TP+FP}$	Positive Predictive Value
F1-Score	$F1 = \frac{2 \times Recall \times Precision}{Recall+Precision}$	Evaluates accuracy of prediction

Using the formulae in Table 5.1, the F1-scores were calculated for different thresholds ranging from 0.05–0.95 in steps of 0.05. As seen in Figure 5.3, the best F1-Score of 0.879, lies between thresholds 0.4–0.45. This range confirmed that the results from Figure 5.1 and 5.2 are correct and corresponds to the threshold with the highest F1-Score in Figure 5.3. With these results, it was decided that if the probability of a student entering the classroom is lower than 0.45, the system should classify that student as unknown.

thresholds	FN	TP	FP	Precision	Recall	F1
0.05>	0	2103	644	0.765562432	1	0.867216
0.05 - 0.1	9	2094	644	0.764791819	0.99572	0.865111
0.1 - 0.15	78	2025	601	0.771134806	0.96291	0.856418
0.15-0.2	181	1922	471	0.80317593	0.913932	0.854982
0.2-0.25	235	1868	354	0.840684068	0.888255	0.863815
0.25-0.3	281	1822	262	0.87428023	0.866381	0.870313
0.3-0.35	327	1776	196	0.900608519	0.844508	0.871656
0.35-0.4	356	1747	148	0.921899736	0.830718	0.873937
0.4-0.45	372	1731	105	0.942810458	0.82311	0.878903
0.45-0.5	410	1693	75	0.957579186	0.80504	0.874709
0.5-0.55	482	1621	57	0.966030989	0.770804	0.857445
0.55-0.6	610	1493	40	0.973907371	0.709938	0.821232
0.6-0.65	803	1300	31	0.976709241	0.618165	0.757135
0.65-0.7	1009	1094	19	0.982929021	0.520209	0.680348
0.7-0.75	1213	890	9	0.989988877	0.423205	0.592938
0.75-0.8	1420	683	3	0.995626822	0.324774	0.489781
0.8-0.85	1705	398	0	1	0.189253	0.318273
0.85-0.9	1938	165	0	1	0.078459	0.145503
0.9-0.95	2080	23	0	1	0.010937	0.021637
MAX						0.878903

Figure 5.3: F1-Score Calculations

5.3 Testing

5.3.1 SVM Testing Results Using Yale Data Set

The Yale data set consists of 42 subjects each containing 65 face images taken at various angles and lighting. 75% of the face samples per subject were used for training and 25% were used for testing. This yielded a test recognition rate of 92.1630% and a test error rate of 7.8370%. The prediction took 0.0009 seconds per sample on average.

5.3.2 Accuracy vs Class Size

Since classes will not always be the same size, the effect of class size had to be tested for its effect on the prediction accuracy of the model. Class sizes were tested ranging from 5–42 subjects in steps of 2. It was expected that as the class size increased the the accuracy would decrease. Although this was the general trend, the difference in accuracys were minute as seen in Figure 5.4.

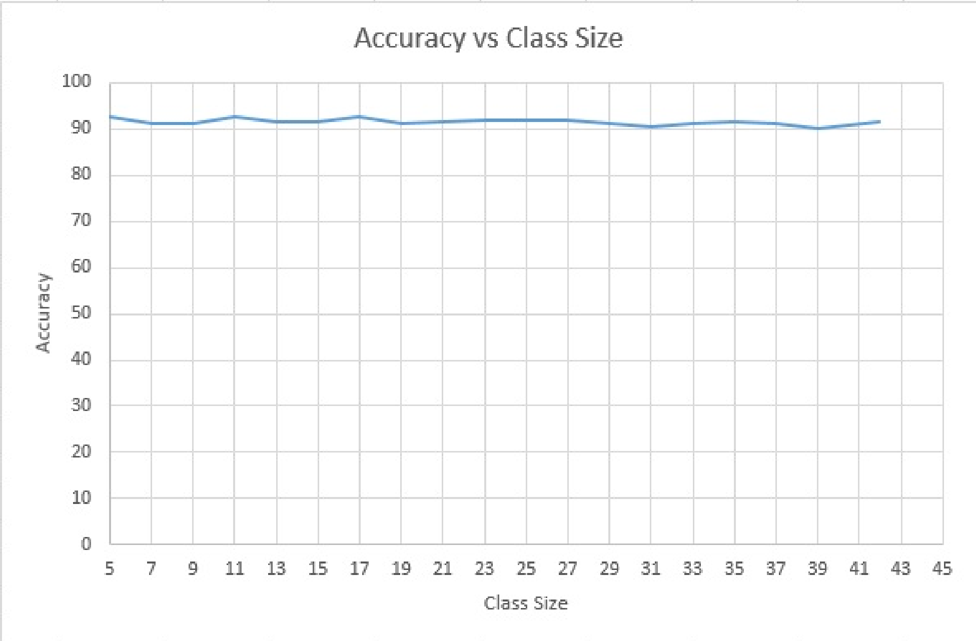

Figure 5.4: Accuracy vs Class Size

5.3.3 Accuracy tests for known and unknown subjects with threshold applied

Table 5.2: Accuracy tests for known and unknown people

Accuracy tests for known and unknown people		
Question	**Result**	**Total No. Subjects**
How many of the registered, known people were above the threshold and correctly predicted?	2287	2471
Of the samples of known people passed in, how many were either below the threshold and incorrectly predicted?	184	2471
Of the unknown, unregistered people, how many were incorrectly above the threshold?	48	469
Of the unknown people, how many were correctly below the threshold?	421	469

5.3.4 Duration of pause to record attendance successfully

As described in Section 5.2.1, attendance will only be recorded in a spreadsheet if the system recognizes the same student for five continuous frames. A slight pause in front of the camera is needed for this verification process to take place. On timing five different subjects on how long the system took to record their attendance it was found that the average duration of the pause was 3.009 seconds and it was decided that the student will be required to pause for at least three seconds for their attendance to be recorded. The results are set out in Table 5.3.

Table 5.3: Duration of pause

Duration of pause	
Subjects	**Time in seconds**
Subject 1	2.83
Subject 2	3.24
Subject 3	3.52
Subject 4	2.89
Subject 5	2.56
Average	**3.009**

Appendix A

User Manual

A.1 Introduction

This chapter discusses usability and system requirements needed to run the Automated Attendance Register system. Figures A.1 to A.4 demonstrates how the system works.

Figure A.1 is the main menu and consists of two buttons: 'Train Face'—which trains a new students face and 'Record Attendance'—which records attendance and creates a register.

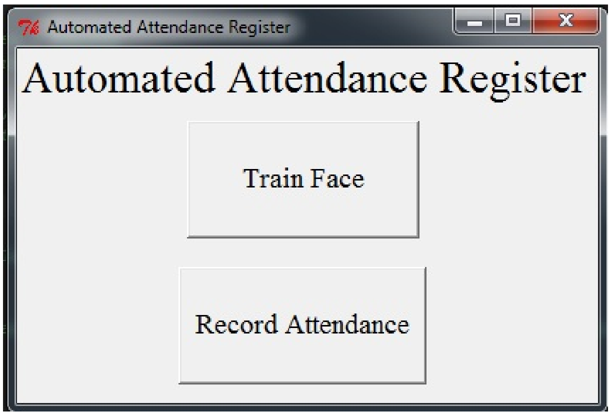

Figure A.1: Main Menu

When 'Train Face' is clicked the system takes the user to Figure A.2, prompting the user to enter the student number, name and surname of the student. When the 'save' button is clicked the system takes the user to figure A.3. The 'back' button takes the user back to the main menu.

Figure A.2: Train Face

As seen in Figure A.3, a live video feed appears prompting the user to press the letter ‘p’ on the keyboard to take a picture or ‘q’ to save and quit. The user is required to take 15–25 different face images at different angles and lighting conditions. The window to the right of the live video stream is the cropped, gray scaled image that will be captured and saved to a directory which can be found in `/face_profiles/`⟨*StudentNumber Name Surname*⟩. When ‘q’is pressed the system returns to the main menu.

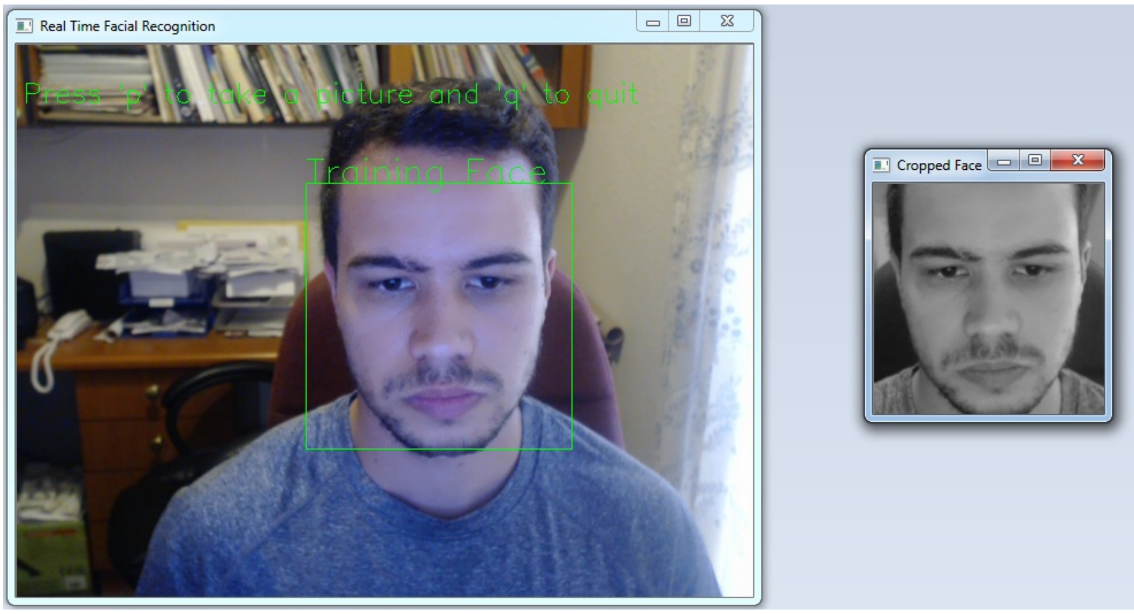

Figure A.3: Taking Pictures

When ‘Train Face’ is clicked the system takes the user to a live video feed as seen in Figure A.4. When the student enters the class, a five second pause is required so that the system can correctly capture and predict the face. The predicted student’s details are displayed on top of the rectangle of the detected face. When attendance is recorded successfully, a pop up window will appear notifying the user. When the ‘OK’ button is clicked, the next student can step into the frame and their attendance can be recorded. When ‘ESC’ or ‘q’ is pressed, the system closes and a register is created named ‘Register.csv’ which can be found in /`scripts/Register.csv`.

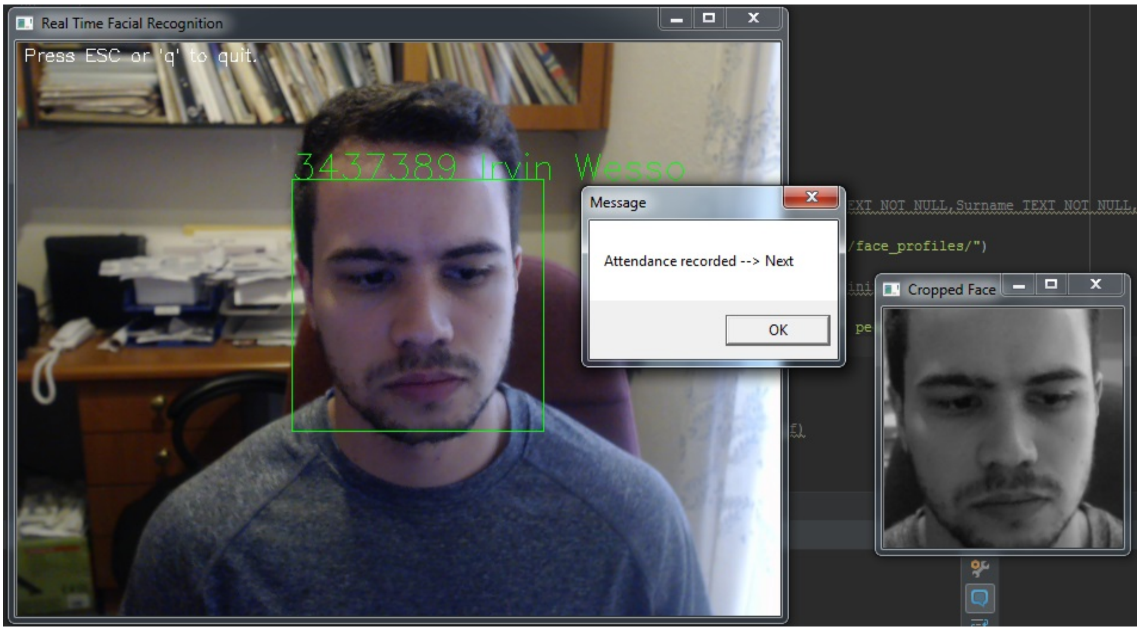

Figure A.4: Record Attendance

A.1.1 System Requirements

- Python 2.7
- OpenCV
- HD WebCam
- Scikit-Image
- scikit-learn
- sklearn
- svm
- tkinter
- matplotlib
- pip
- Windows 7 x86
- Note: PyCharm on windows makes these libraries easier to install.

A.1.2 Instructions

Download 'Automated Attendance Register.zip' from 'cs.uwc.ac.za/∼iwesso' under 'Term 4'. Unzip the folder and ensure all the above libraries are installed by typing 'python -m pip list'. Open CMD as administrator and navigate to the folder that contains 'frontEnd.py'. In the terminal, type 'python frontEnd.py' to run the system. The main menu in Figure A.1 should appear.

A.1.3 Contact Details

For questions, feedback or advice feel free to send me an email on 3437389@myuwc.ac.za.

Appendix B

Source Code

B.1 Introduction

The source code for implementing the Automated Attendance Register system is found in this chapter.

B.1.1 Utilities

```

import cv2
import numpy as np
from scipy import ndimage
import os
import errno
import sys
import logging
import shutil


# ################################################################################
# Used For Facial Tracking and Traning in OpenCV

def read_images_from_single_face_profile(face_profile, face_profile_name_index, dim = (50, 50)):
    """
    Reads all the images from one specified face profile into ndarrays

    Parameters
    ----------
    face_profile: string
```

```
        The directory path of a specified face profile

    face_profile_name_index: int
        The name corresponding to the face profile is encoded in its index

    dim: tuple = (int, int)
        The new dimensions of the images to resize to

    Returns
    --------
    X_data : numpy array, shape = (number_of_faces_in_one_face_profile, face_pixel_width * face_pixel_height)
        A face data array contains the face image pixel rgb values of all the images in the specified face profile

    Y_data : numpy array, shape = (number_of_images_in_face_profiles, 1)
        A face_profile_index data array contains the index of the face profile name of the specified face profile directory

    """
    X_data = np.array([])
    index = 0
    for the_file in os.listdir(face_profile):
        file_path = os.path.join(face_profile, the_file)
        if file_path.endswith(".png") or file_path.endswith(".jpg") or file_path.endswith(".jpeg") or file_path.endswith(".pgm"):
            img = cv2.imread(file_path, 0)
            img = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
            img_data = img.ravel()
            X_data = img_data if not X_data.shape[0] else np.vstack((X_data,img_data))
            index += 1

    if index == 0 :
        shutil.rmtree(face_profile)
```

```
        logging.error("\nThere exists face profiles without images")

    Y_data = np.empty(index, dtype = int)
    Y_data.fill(face_profile_name_index)
    return X_data, Y_data

def delete_empty_profile(face_profile_directory):
    """
    Deletes empty face profiles in face profile directory and logs error if face profiles contain too little images

    Parameters
    ----------
    face_profile_directory: string
        The directory path of the specified face profile directory

    """
    for face_profile in os.listdir(face_profile_directory):
        if "." not in str(face_profile):
            face_profile = os.path.join(face_profile_directory, face_profile)
            index = 0
            for the_file in os.listdir(face_profile):
                file_path = os.path.join(face_profile, the_file)
                if file_path.endswith(".png") or file_path.endswith(".jpg") or file_path.endswith(".jpeg") or file_path.endswith(".pgm"):
                    index += 1
            if index == 0 :
                shutil.rmtree(face_profile)
                print ("\nDeleted ", face_profile, " because it contains no images")
            if index < 2 :
                logging.error("\nFace profile " + str(face_profile) + " contains too little images (At least 2 images are needed)")


def load_training_data(face_profile_directory):
    """
```

```
    Loads all the images from the face profile directory into ndarrays

    Parameters
    ----------
    face_profile_directory: string
        The directory path of the specified face profile directory

    face_profile_names: list
        The index corresponding to the names corresponding to the face profile directory

    Returns
    -------
    X_data : numpy array, shape = (number_of_faces_in_face_profiles, face_pixel_width * face_pixel_height)
        A face data array contains the face image pixel rgb values of all face_profiles

    Y_data : numpy array, shape = (number_of_face_profiles, 1)
        A face_profile_index data array contains the indexs of all the face profile names

    """
    delete_empty_profile(face_profile_directory)  # delete profile directory without images

    # Get a the list of folder names in face_profile as the profile names
    face_profile_names = [d for d in os.listdir(face_profile_directory) if "." not in str(d)]

    if len(face_profile_names) < 2:
        logging.error("\nFace profile contains too little profiles (At least 2 profiles are needed)")
        exit()

    first_data = str(face_profile_names[0])
    first_data_path = os.path.join(face_profile_directory, first_data)
```

```
    X1, y1 = read_images_from_single_face_profile(first_data_path, 0)
    X_data = X1
    Y_data = y1
    print ("Loading Database: ")
    print (0, "    ",X1.shape[0]," images are loaded from:", first_data_path)
    for i in range(1, len(face_profile_names)):
        directory_name = str(face_profile_names[i])
        directory_path = os.path.join(face_profile_directory, directory_name)
        tempX, tempY = read_images_from_single_face_profile(directory_path, i)
        X_data = np.concatenate((X_data, tempX), axis=0)
        Y_data = np.append(Y_data, tempY)
        print (i, "    ",tempX.shape[0]," images are loaded from:", directory_path)

    return X_data, Y_data, face_profile_names


def rotate_image(img, rotation, scale = 1.0):
    """
    Rotate an image rgb matrix with the same dimensions

    Parameters
    ----------
    image: string
        the image rgb matrix

    rotation: int
        The rotation angle in which the image rotates to

    scale: float
        The scale multiplier of the rotated image

    Returns
    -------
    rot_img : numpy array
        Rotated image after rotation

```

```
    """
    if rotation == 0: return img
    h, w = img.shape[:2]
    rot_mat = cv2.getRotationMatrix2D((w/2, h/2), rotation, scale)
    rot_img = cv2.warpAffine(img, rot_mat, (w, h), flags=cv2.
        INTER_LINEAR)
    return rot_img

def trim(img, dim):
    """
    Trim the four sides(black paddings) of the image matrix and
        crop out the middle with a new dimension

    Parameters
    ----------
    img: string
        the image rgb matrix

    dim: tuple (int, int)
        The new dimen the image is trimmed to

    Returns
    -------
    trimmed_img : numpy array
        The trimmed image after removing black paddings from four
            sides

    """

    # if the img has a smaller dimension then return the origin
        image
    if dim[1] >= img.shape[0] and dim[0] >= img.shape[1]: return
        img
    x = int((img.shape[0] - dim[1])/2) + 1
    y = int((img.shape[1] - dim[0])/2) + 1
    trimmed_img = img[x: x + dim[1], y: y + dim[0]]    # crop the
        image
    return trimmed_img
```

```
def clean_directory(face_profile):
    """
    Deletes all the files in the specified face profile

    Parameters
    ----------
    face_profile: string
        The directory path of a specified face profile

    """

    for the_file in os.listdir(face_profile):
        file_path = os.path.join(face_profile, the_file)
        try:
            if os.path.isfile(file_path):
                os.unlink(file_path)
            #elif os.path.isdir(file_path): shutil.rmtree(
                file_path)
        except Exception as e:
            print (e)


def create_directory(face_profile):
    """
    Create a face profile directory for saving images

    Parameters
    ----------
    face_profile: string
        The directory path of a specified face profile

    """
    try:
        print ("Making directory")
        os.makedirs(face_profile)
    except OSError as exception:
        if exception.errno != errno.EEXIST:
            print ("The specified face profile already existed, it
                will be override")
            raise

```

```
def create_profile_in_database(face_profile_name, database_path="../face_profiles/", clean_directory=False):
    """
    Create a face profile directory in the database

    Parameters
    ----------
    face_profile_name: string
        The specified face profile name of a specified face profile folder

    database_path: string
        Default database directory

    clean_directory: boolean
        Clean the directory if the user already exists

    Returns
    -------
    face_profile_path: string
        The path of the face profile created

    """
    face_profile_path = database_path + face_profile_name + "/"
    create_directory(face_profile_path)
    # Delete all the pictures before recording new
    if clean_directory:
        clean_directory(face_profile_path)
    return face_profile_path
```

B.1.2 Building the SVM

```

import cv2
import os
import numpy as np
from scipy import ndimage
from time import time
import warnings

with warnings.catch_warnings():
    warnings.simplefilter("ignore")
```

```
    from sklearn.cross_validation import train_test_split

from sklearn.datasets import fetch_lfw_people
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import RandomizedPCA
from sklearn.svm import SVC
import matplotlib.pyplot as plt
import numpy
import utils as ut
from sklearn.metrics import accuracy_score


def test_SVM(face_profile_data, face_profile_name_index, face_dim,
        face_profile_names):
    """
    Testing: Build the SVM classification modle using the
        face_profile_data matrix (numOfFace X numOfPixel) and
        face_profile_name_index array, face_dim is a tuple of the
        dimension of each image(h,w) Returns the SVM
        classification modle
    Parameters
    ----------
    face_profile_data : ndarray (number_of_images_in_face_profiles
        , width * height of the image)
        The pca that contains the top eigenvectors extracted using
            approximated Singular Value Decomposition of the data

    face_profile_name_index : ndarray
        The name corresponding to the face profile is encoded in
            its index

    face_dim : tuple (int, int)
        The dimension of the face data is reshaped to

    face_profile_names: ndarray
        The names corresponding to the face profiles
    Returns
    -------
    clf : theano object
```

```
        The trained SVM classification model

    pca : theano ojbect
        The pca that contains the top 150 eigenvectors extracted using approximated Singular Value Decomposition of the data

    """
    X = face_profile_data
    y = face_profile_name_index

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

    # Compute a PCA (eigenfaces) on the face dataset (treated as unlabeled
    # dataset): unsupervised feature extraction / dimensionality reduction
    n_components = 150 # maximum number of components to keep

    print("\nExtracting the top %d eigenfaces from %d faces" % (n_components, X_train.shape[0]))

    pca = RandomizedPCA(n_components=n_components, whiten=True).fit(X_train)
    eigenfaces = pca.components_.reshape((n_components, face_dim[0], face_dim[1]))


    # This portion of the code is used if the data is scarce, it uses the number
    # of imputs as the number of features
    # pca = RandomizedPCA(n_components=None, whiten=True).fit(X_train)
    # eigenfaces = pca.components_.reshape((pca.components_.shape[0], face_dim[0], face_dim[1]))

    print("\nProjecting the input data on the eigenfaces orthonormal basis")
    X_train_pca = pca.transform(X_train)
    X_test_pca = pca.transform(X_test)
```

```

    # Train a SVM classification model

    print("\nFitting the classifier to the training set")
    param_grid = {'C': [1e3, 5e3, 1e4, 5e4, 1e5],
                  'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1], }



    # Best Estimator found using Radial Basis Function Kernal:
    clf = SVC(C=1000.0, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape=None, degree=3, gamma=0.0001, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)


    # ###############################################################################

    # Quantitative evaluation of the model quality on the test set
    print("\nPredicting people's names on the test set")
    t0 = time()
    y_pred = clf.predict(X_test_pca)
    print("\nPrediction took %0.8f second per sample on average" % ((time() - t0)/y_pred.shape[0]*1.0))

    # print "predicated names: ", y_pred
    # print "actual names: ", y_test
    error_rate = errorRate(y_pred, y_test)
    print ("\nTest Error Rate: %0.4f %%" % (error_rate * 100))
    print ("Test Recognition Rate: %0.4f %%" % ((1.0 - error_rate) * 100))


    return clf, pca


```

```
def plot_gallery(images, titles, face_dim, n_row=3, n_col=4):
#    """Helper function to plot a gallery of portraits"""
        plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
        plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
        for i in range(n_row * n_col):
            plt.subplot(n_row, n_col, i + 1)
            plt.imshow(images[i].reshape(face_dim), cmap=plt.cm.gray)
            plt.title(titles[i], size=12)
            plt.xticks(())
            plt.yticks(())


def build_SVC(face_profile_data, face_profile_name_index, face_dim):
    """
    Build the SVM classification modle using the face_profile_data matrix (numOfFace X numOfPixel) and face_profile_name_index array, face_dim is a tuple of the dimension of each image(h,w) Returns the SVM classification modle
    Parameters
    ----------
    face_profile_data : ndarray (number_of_images_in_face_profiles, width * height of the image)
        The pca that contains the top eigenvectors extracted using approximated Singular Value Decomposition of the data

    face_profile_name_index : ndarray
        The name corresponding to the face profile is encoded in its index

    face_dim : tuple (int, int)
        The dimension of the face data is reshaped to

    Returns
    -------
    clf : theano object
        The trained SVM classification model

```

```
    pca : theano ojbect
        The pca that contains the top 150 eigenvectors extracted using approximated Singular Value Decomposition of the data

    """

    X = face_profile_data
    y = face_profile_name_index

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

    # Compute a PCA (eigenfaces) on the face dataset (treated as unlabeled
    # dataset): unsupervised feature extraction / dimensionality reduction
    n_components = 150 # maximum number of components to keep

    print("\nExtracting the top %d eigenfaces from %d faces" % (n_components, X_train.shape[0]))

    pca = RandomizedPCA(n_components=n_components, whiten=True).fit(X_train)
    eigenfaces = pca.components_.reshape((n_components, face_dim[0], face_dim[1]))
    eigenface_titles = ["eigenface %d" % i for i in range(eigenfaces.shape[0])]
    plot_gallery(eigenfaces, eigenface_titles, face_dim)



    print("\nProjecting the input data on the eigenfaces orthonormal basis")
    X_train_pca = pca.transform(X_train)
    X_test_pca = pca.transform(X_test)

    # Train a SVM classification model

    print("\nFitting the classifier to the training set")
    param_grid = {'C': [1e3, 5e3, 1e4, 5e4, 1e5],
```

```
                    'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01,
                        0.1], }
    # clf = GridSearchCV(SVC(kernel='rbf', class_weight='balanced
        '), param_grid)

    # Best Estimator found using Radial Basis Function Kernal:
    clf = SVC(C=1000.0, cache_size=200, class_weight='balanced',
        coef0=0.0,
  decision_function_shape=None, degree=3, gamma=0.0001, kernel='
      rbf',
  max_iter=-1, probability=True, random_state=None, shrinking=True
      ,
  tol=0.001, verbose=False)
    # Train_pca with  Test Error Rate:  0.088424437299
    # Train_pca with  Test Recognition Rate:  0.911575562701

    clf = clf.fit(X_train_pca, y_train)
    # print("\nBest estimator found by grid search:")
    # print(clf.best_estimator_)

    #
        ################################################################################

    # Quantitative evaluation of the model quality on the test set
    print("\nPredicting people's names on the test set")
    t0 = time()
    y_pred = clf.predict(X_test_pca)
    probs = clf.predict_proba(X_test_pca)

    print("\nPrediction took %s per sample on average" % ((time()
        - t0)/y_pred.shape[0]*1.0))

    error_rate = errorRate(y_pred, y_test)
    print ("\nTest Error Rate: %0.4f %%" % (error_rate * 100))
    print ("Test Recognition Rate: %0.4f %%" % ((1.0 - error_rate)
        * 100))

    return clf, pca


def predict(clf, pca, img, face_profile_names):
```

```
    """
    Predict the name of the supplied image from the list of face
        profile names

    Parameters
    ----------
    clf: theano object
        The trained svm classifier

    pca: theano object
        The pca that contains the top eigenvectors extracted using
            approximated Singular Value Decomposition of the data

    img: ndarray
        The input image for prediction

    face_profile_names: list
       The names corresponding to the face profiles
    Returns
    -------
    name : string
        The predicated name

    """


    img = img.ravel()

    img=img.reshape(1, -1)

    # Apply dimentionality reduction on img, img is projected on
        the first principal components previous extracted from the
         Yale Extended dataset B.
    principle_components = pca.transform(img)

    pred = clf.predict(principle_components)
    probs = clf.predict_proba(principle_components)[0]
  # print(max(probs))
    if(max(probs)<0.4):
        name ="Unknown"
    else:
```

```
            str = pred[0]
            name = face_profile_names[str]

    return name

def errorRate(pred, actual):
    """
    Calculate name prediction error rate

    Parameters
    ----------
    pred: ndarray (1, number_of_images_in_face_profiles)
        The predicated names of the test dataset

    actual: ndarray (1, number_of_images_in_face_profiles)
        The actual names of the test dataset

    Returns
    -------
    error_rate: float
        The calcualted error rate

    """
    if pred.shape != actual.shape: return None
    error_rate = np.count_nonzero(pred - actual)/float(pred.shape[0])
    return error_rate
```

B.1.3 Training

```
"""
====================================================
Automated Attendance Register
====================================================

The dataset used is the Extended Yale Database B Cropped

  http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html


Summary:
        Used for face profile data collection in real time
```

```
        face training for recognition

Run:
    * Training for face recognition using the command below. face_profile_name is the name of the user face profile directory that you want to create in the default ../face_profiles/ folder for storing user face images and training the SVM classification model:

        python train.py [face_profile_name=<the name of the profile folder in database>]

    * Example to create a face profile named David:

        python train.py David


Usage during run time:

        press and hold 'p' to take pictures of you continuously once a cropped face is detected from a pop up window. All images are saved under ../face_profiles/face_profile_name

        press 'q' or 'ESC' to quit the application


"""

import cv2
import numpy as np
from scipy import ndimage
import sys
import os
from StringIO import StringIO
import utils as ut
from Tkinter import *



FACE_DIM = (200, 200)
```

```python
SKIP_FRAME = 2          # the fixed skip frame
frame_skip_rate = 0 # skip SKIP_FRAME frames every other frame
SCALE_FACTOR = 1 # used to resize the captured frame for face detection for faster processing speed
face_cascade = cv2.CascadeClassifier("../classifier/haarcascade_frontalface_default.xml") #create a cascade classifier
sideFace_cascade = cv2.CascadeClassifier('../classifier/haarcascade_profileface.xml')

# dictionary mapping used to keep track of head rotation maps
rotation_maps = {
    "left": np.array([-30, 0, 30]),
    "right": np.array([30, 0, -30]),
    "middle": np.array([0, -30, 30]),
}

def get_rotation_map(rotation):
    """ Takes in an angle rotation, and returns an optimized rotation map """
    if rotation > 0: return rotation_maps.get("right", None)
    if rotation < 0: return rotation_maps.get("left", None)
    if rotation == 0: return rotation_maps.get("middle", None)

current_rotation_map = get_rotation_map(0)


webcam = cv2.VideoCapture(0)
webcam.set(cv2.CAP_PROP_FPS,60)
ret, frame = webcam.read() # get first frame
frame_scale = (frame.shape[1]/SCALE_FACTOR,frame.shape[0]/SCALE_FACTOR)  # (y, x)

cropped_face = []
num_of_face_to_collect = 150
num_of_face_saved = 0

#  For saving face data to directory
profile_folder_path = None

window2 = Tk()
```

```
filename=StringVar()
window2.title("Automated Attendance Register")
label1 = Label(text="Enter student number followed by name and
    surname", font=("Times New Roman", 15))
label1.grid(column=0, row=0)

entry1 = Entry(textvariable=filename)
entry1.grid(column=0, row=1)

def back():
    window2.withdraw()
    os.system("python frontEnd.py")

def save():
    global filename
    filename=entry1.get()
    window2.destroy()


btnBack = Button(window2, text="Back", font=("Times New Roman",
    11), command=back)
btnBack.grid(row=2, column=1, padx=5)

btnSave = Button(window2, text="Save", font=("Times New Roman",
    11), command=save)
btnSave.grid(row=2, column=0, padx=5)

window2.mainloop()

#print "Enter student number followed by name and surname"

print(filename)

profile_folder_path = ut.create_profile_in_database(filename)


while ret:
    key = cv2.waitKey(1)
    # exit on 'q' 'esc' 'Q'
    if key in [27, ord('Q'), ord('q')]:
        break
```

```python
    # resize the captured frame for face detection to increase processing speed
    resized_frame = cv2.resize(frame, frame_scale)

    processed_frame = resized_frame
    # Skip a frame if the no face was found last frame
    if frame_skip_rate == 0:
        faceFound = False
        for rotation in current_rotation_map:

            rotated_frame = ndimage.rotate(resized_frame, rotation)

            gray = cv2.cvtColor(rotated_frame, cv2.COLOR_BGR2GRAY)

            # return tuple is empty, ndarray if detected face
            faces = face_cascade.detectMultiScale(
                gray,
                scaleFactor=1.3,
                minNeighbors=5,
                minSize=(30, 30),
                flags=cv2.CASCADE_SCALE_IMAGE
            )

            # If frontal face detector failed, use profileface detector
            faces = faces if len(faces) else sideFace_cascade.detectMultiScale(
                gray,
                scaleFactor=1.3,
                minNeighbors=5,
                minSize=(30, 30),
                flags=cv2.CASCADE_SCALE_IMAGE
            )

            if len(faces):
                for f in faces:
                    x, y, w, h = [ v for v in f ] # scale the bounding box back to original frame size
                    cropped_face = rotated_frame[y: y + h, x: x + w]    # img[y: y + h, x: x + w]
```

```
                cropped_face = cv2.resize(cropped_face, FACE_DIM, interpolation = cv2.INTER_AREA)
                cv2.rectangle(rotated_frame, (x,y), (x+w,y+h), (0,255,0))
                cv2.putText(rotated_frame, "Training Face", (x,y), cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0,255,0))

            # rotate the frame back and trim the black paddings
            processed_frame = ut.trim(ut.rotate_image(rotated_frame, rotation * (-1)), frame_scale)

            # reset the optmized rotation map
            current_rotation_map = get_rotation_map(rotation)

            faceFound = True


            break

    if faceFound:
        frame_skip_rate = 0
        # print "Face Found"
    else:
        frame_skip_rate = SKIP_FRAME
        # print "Face Not Found"

else:
    frame_skip_rate -= 1
    # print "Face Not Found"



cv2.putText(processed_frame, "Press 'p' to take a picture and 'q' to quit", (5, 50),
        cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,255,0))

cv2.imshow("Real Time Facial Recognition", processed_frame)


```

```

    if len(cropped_face):
        cv2.imshow("Cropped Face", cv2.cvtColor(cropped_face, cv2.COLOR_BGR2GRAY))
        if num_of_face_saved < num_of_face_to_collect and key == ord('p'):
            face_to_save = cv2.resize(cropped_face, (50, 50), interpolation = cv2.INTER_AREA)
            face_name = profile_folder_path+str(num_of_face_saved)+".png"
            save=cv2.cvtColor(face_to_save, cv2.COLOR_BGR2GRAY)
            cv2.imwrite(face_name, save)
            print "Pic Saved: ", face_name
            num_of_face_saved += 1

    # get next frame
    ret, frame = webcam.read()


webcam.release()
cv2.destroyAllWindows()
os.system("python frontEnd.py")
```

Bibliography

Bishop, C. M. (2016). *Pattern Recognition and Machine Learning*. Springer-Verlag New York.

Chintalapati, S. and Raghunadh, M. V. (2013). Automated attendance management system based on face recognition algorithms. In *IEEE International Conference on Computational Intelligence and Computing Research*, pages 1–5.

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87.

Jensen, O. H. (2008). Implementing the Viola-Jones face detection algorithm. Master's thesis, Technical University of Denmark, Lyngby, Denmark.

Jha, A. (2007). Class room attendance system using facial recognition system. *International Journal of Mathematics, Science, Technology and Management*, 2(3):4–7.

Kumar, K. S., Semwal, V. B., and Tripathi, R. C. (2011). Real time face recognition using adaboost improved fast PCA algorithm. *International Journal of Artificial Intelligence and Applications*, 2(3):45–58.

Newman-Ford, L., Fitzgibbon, K., Lloyd, S., and Thomas, S. (2008). A large-scale investigation into the relationship between attendance and attainment: a study using an innovative, electronic attendance monitoring system. *Studies in Higher Education*, 33(6):699–717.

Noble, W. S. (2006). What is a support vector machine? *Nature Biotechnology*, 24(12):15–65.

Othman, M., Ismail, S. N., and Raus, M. I. M. (2009). The development of the web-based attendance register system (ARS) for higher academic institution: From feasibility study to the design phase. *International Journal of Computer Science and Network Security*, 9(10):203–208.

Patel, U. A. and Priya, S. (2014). Development of a student attendance management system using RFID and face recognition: A review. *International Journal of Advance Research in Computer Science and Management Studies*, 2(8):109–119.

Turk, M. A. and Pentland, A. P. (1991). Face recognition using eigenfaces. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, pages 586–591.

Wang, Y.-Q. (2014). An analysis of the Viola-Jones face detection algorithm. *Image Processing On Line*, 4:128–148.