Project No: SCSE19-0352

NANYANG TECHNOLOGICAL UNIVERSITY

Rapid Facial Recognition

Through Wearable Cameras

Submitted by: Sim Jun Kai

Matriculation Number: U1721530F

Supervisor: CHAM, Tat Jen

Co-supervisor: -

Project No: SCSE19-0352

School of Computer Engineering

A final year project report presented to the Nanyang Technological University

in partial fulfilment of the requirements of the degree of

Bachelor of Engineering

2020

Table of Contents

Abstract

Face Recognition has been one of the most popular topics in the industry over the past decades. It is a biometric software that has many creative usages, for example, the camera of a smartphone where it will automatically focus on the face of the person and in China, cameras are used to capture the face of those people who jaywalk. Other than those usages of face recognition system, it can be served as a memory aid and provides helpful assistance for the people who have memory disabilities.

While the technology is still developing, researchers have come out with algorithms and techniques to improvise the existing performance of face recognition system. The new improved face recognition system allows the machine to learn and identify correctly from a massive amount of training sets within a short time frame. This extraordinary leap has made face recognition system simplify and possibly be embedded into our daily life.

The goal of this report is to present a face recognition system that makes use of k-Nearest Neighbors to achieve a rapid recognition of everyone that appears on the screen. Moreover, it can also be used as a memory aid for the users. The report provides a detailed explanation of the software used, these include face recognition API, python libraries, pre-trained model and the reasons for choosing such techniques and methods to achieve the goals of the project. In addition, the flow chart and decision tree of the program will be used to provide a better illustration of how the face recognition system works. Lastly, the report has also stated further improvements which allow the whole face recognition project to achieve better user satisfaction and performance enhancement of the system.

Acknowledgements

I would like to express my sincere thanks and great gratitude to my Final Year Project (FYP) supervisor, Professor Cham Tat Jen, for giving me this chance of carrying out his research topic. This topic taught me more than just the knowledge of Machine Learning, Artificial Intelligence, and Face Recognition. It also allows me to have hands-on experience in developing a face recognition application. Also, I really appreciate his guidance on how a research topic should go about and the valuable feedback that he gave.

Jun Kai
March 2020

Acronyms

CCTV	Closed-Circuit Television
GUI	Graphical User Interface
HOG	Histogram of Oriented Gradients
k-NN	K-Nearest Neighbor
SVM	Support Vector Machine
MLP	Multilayer Perceptron
ResNet	Residual Networks
CNN	Convolutional Neural Network
VGG	Visual Geometry Group
LFW	Labeled Faces in the Wild

List of Figures

Figure 27. Predicting with model

Figure 28. Predict Classes and Removing Classification aren't within the threshold

Chapter 1
Introduction

This chapter documents the introduction of the Final Year Project (FYP). This project is for exploring a rapid face recognition system as a facial memory aid for users. The use scenario is as of when a user may have difficulty remembering the names of acquaintances to whom he/she has been previously introduced. Unlike existing face recognition systems, the primary focus of this project is the recognition of a limited number of individual quickly, and with the ability to classify unknown persons as strangers.

1.1 Background

The facial recognition system is a biometric software application capable of uniquely identify or verify a person by comparing and analyzing patterns based on their facial contours. As technologies have been improving rapidly throughout the past few years, using the computer to recognize faces has become a major research area and this results in having the facial recognition system developed rapidly. The most common hardware that uses facial recognition is closed-circuit television (CCTV), for instance. The power capability of facial recognition can be used to search for known criminals or drug offenders and notify the relevant authorities immediately when in the presence of known terrorists or suspected criminals. Furthermore, its variety usage includes tracking down lost children or even missing persons [1].

Throughout the years, facial recognition has made extraordinary leaps in the field and this favorable progress is due to the new improved network designs, techniques, and algorithms [2]. They were all improved and enhanced to achieve a more desirable result. The new improved network design allows facial recognition system to learn

and identify correctly from the massive amount of training sets within a short amount of time. Furthermore, the facial recognition system is simplified enough to allow us to embed it into our daily life. One such example is our daily essential item: Smartphone camera.

1.2 Objectives and Scope

As mention in the ***Introduction***, the objective of the project is to be able to perform a rapid face recognition of an acquaintance through a camera and distinguishing between known acquaintances and strangers. Due to the project requires the user to collect photos of their acquaintances to obtain the necessary dataset needed for face recognition, as such, the project has a few constraints that must be considered.

First, how a program recognizes between strangers and known acquaintances. Second, methods in collecting photos of acquaintances. Lastly, the accuracy of the face recognition and the size of the dataset in order to return the most accurate result. The consideration between the accuracy and size of the dataset is a vital factor in the project because of the objective of the project.

Chapter 2
Material/Equipment Resources and Costing

The project requires the following resources and equipment for one to follow and carry out the project if desired.

2.1 Hardware

The hardware needed for the project is a laptop that has a camera function. The used laptop in this project is **ASUS ZenBook 14**. The webcam model of the laptop is **USB 2.0 HD IR UVC Webcam**. The operating system of the laptop is **Windows 10 Home** and the processor of the laptop is **Intel® Core™ i7-8565U CPU 8th Gen with 16.0 GB Ram installed**.

2.2 Software

There are a few software and libraries used in the project. These are the following programming language, libraries, and downloads required for the successful run of the project.

2.2.1 Sublime

The project uses Sublime Text as the source code editor. It is a cross-platform source code editor with a Python application programming interface.

2.2.2 Python

The programming language used in the project is Python, with a version of 3.7.3. Python has offered many powerful libraries that speed up the development time of the project, thus taking considerably less amount of time to build something in Python than any other language.

2.2.2.1 Anaconda

The project uses Anaconda, with a version of 4.7.11. Anaconda brings convenience to the user because many tools used in data science and machine learning can be installed with one installation. It also creates an independent environment to isolate the problem of having different libraries and versions in one laptop. In addition, Anaconda has useful interaction with pip that allows the user to install additional libraries that are unavailable in the Anaconda package manager [3].

2.2.2.2 Scikit-learn

The project uses Scikit-learn, a free software machine learning library for the Python programming language. It has many classifications, regression and clustering algorithms in-built and in this project, k-nearest neighbour will be used. Scikit-learn is also designed to interoperate with the Python numerical library NumPy which is another library used for this project.

2.2.2.3 OpenCV

The project uses OpenCV, with a version of 3.4.1. OpenCV is a library of programming function aimed to solve real-time computer vision problems. It is mainly used for image processing and in this project, it is used for live webcam and processing captured screenshot

2.2.2.4 PySimpleGUI

The project uses PySimpleGUI, with a version of 4.16.0. PySimpleGUI has a simple concept when it comes to the creation of Graphical User Interface (GUI) with Python programming language, furthermore, its incorporation of OpenCV with GUI in the project is much easier than another GUI library, for example, Tkinter [4].

2.2.2.5 Python Pickle

The project uses Python pickle, with a version of 4.0. In this project, Python pickle is used for the serializing and de-serializing of k-nearest neighbour classifier data stream. The stored data will be saved in the current directory.

2.2.2.6 Face Recognition API

The project uses facial recognition API for Python for face recognition. It is known to be the world's simplest face recognition library that recognizes and manipulates faces from Python. It is built using Dlib and the model has an accuracy of 99.38% on the Labelled Faces in the Wild Benchmark. In addition, it also comprises of many in-built features that help the project to progress faster.

2.2.2.7 Dlib

The project uses Dlib, with a version of 19.18.0. Dlib is a toolkit for making real-world machine learning and data analysis applications for C++. However, it has good and ease of use in Python bindings. In this project, Dlib provides a simple frontal face detector and pre-trained models for the ease of face recognition.

2.2.2.8 CMake

The project requires CMake, with a version of 3.15.3 installed before installing of Dlib. CMake is a free and open-source software tools for managing the build process of software using a compiler-independent method. It is used to generate make files for OpenCV and a requirement for the successful installation of Dlib.

2.2.2.9 Python OS

The project uses Python os module for the reading and writing to the directory.

2.2.2.10 Python Numpy

The project does not use Numpy in the coding portion. However, Numpy is needed to be downloaded for the other successful runs of libraries; for example, Dlib and OpenCV.

2.2.2.11 Python math

The project uses math modules for the calculation of mathematical function in the k-nearest neighbors.

2.3 Project Schedule

The project schedule is illustrated in the following Figure.

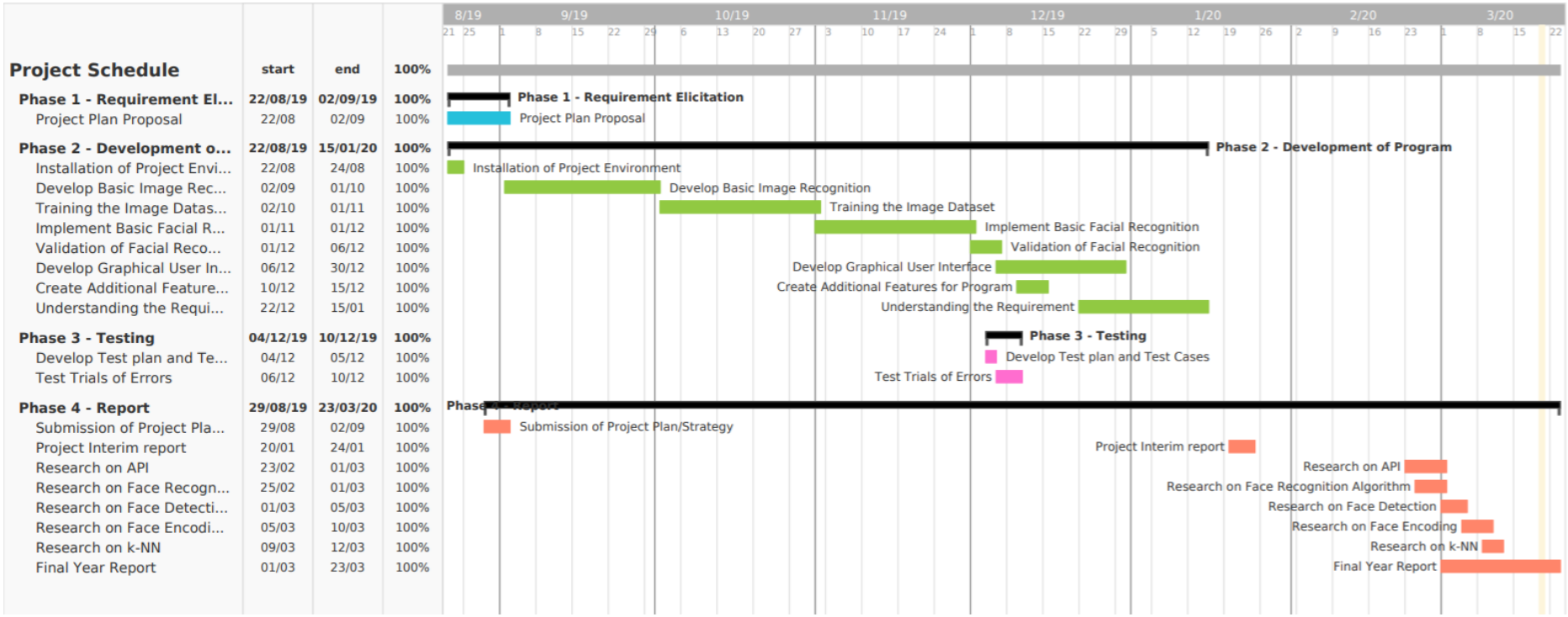

Figure 1. Project Schedule

2.4 Report Organization

This report is organized into the following chapters:

1. Introduction
    - This chapter includes the background of the project, specifies the objectives and the scope of the work.
2. Material/Equipment Resources and Costing
    - This chapter introduces the resources used for implementing the application.
3. Project Schedule
    - This chapter describes the timelines of the overall project and the key milestones faced during the period.
4. Report Organization
    - This chapter describes the report organization of the overall project.
5. Project Review (GUI)
    - This chapter discusses how the program implements the graphical user interface to add training data and turning on the laptop's camera.
6. Project Review (Train)
    - This chapter discusses the algorithm and method of Face detection, Face encoding used for the program.
7. Project Review (Predict)
    - This chapter discusses how the program makes use of a classifier to predict an input face.
8. Future Improvement and Recommendation
    - This chapter provides future enhancement recommendations for the project.
9. Conclusion
    - This chapter gives an overall conclusion of the project.
10. References
    - This chapter shows all the citations and references from reports.

Chapter 3

Program Review (GUI)

3.1 Graphical User Interface

The program has a Graphical User Interface (GUI) built to provide a better user experience. It consists of displaying the name, live camera, and an Add button. To exit the program, press the Close button.

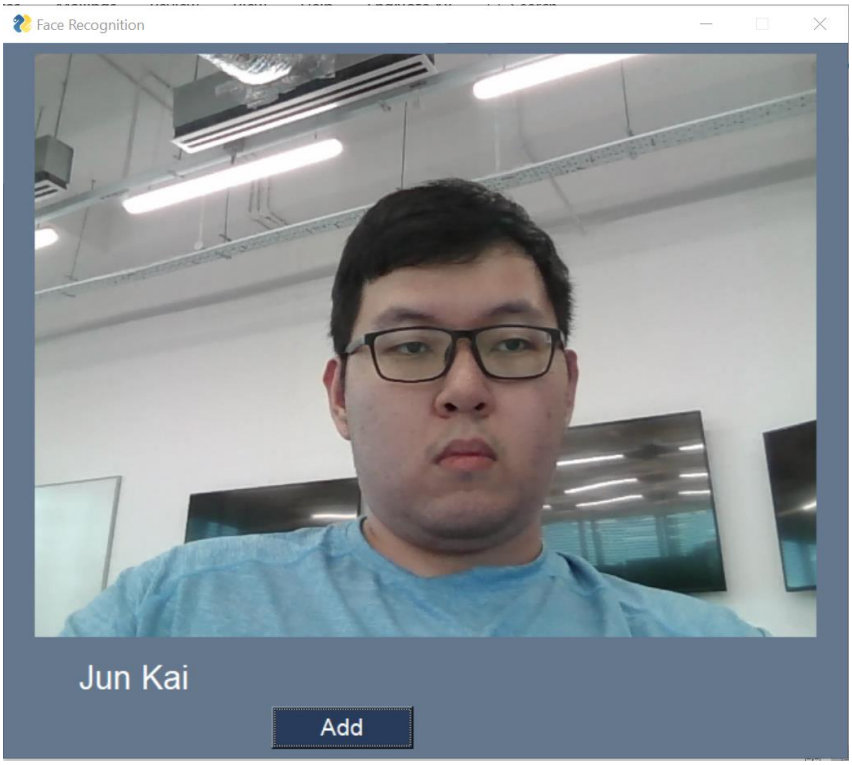

Figure 2. Graphical User Interface of the Program

3.1.1 Add Training Data

The program will prompt a pop-up textbox for the user to enter. If the user enters an existing name, an error message will be prompted (*Refer to Figure 3*). Upon successful entry, the program will proceed to create an image folder for the respective name and take a total of 10 screenshots *(Refer to Figure 4 and Figure 5)*.

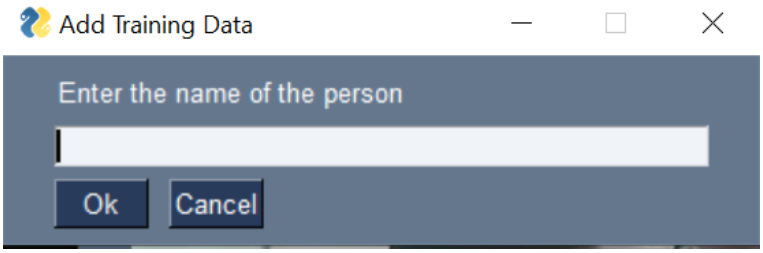

Figure 3. Pop-up textbox for adding user

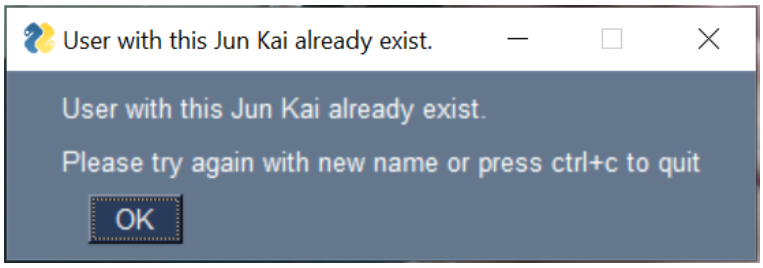

Figure 4. Error Message when existing user occur

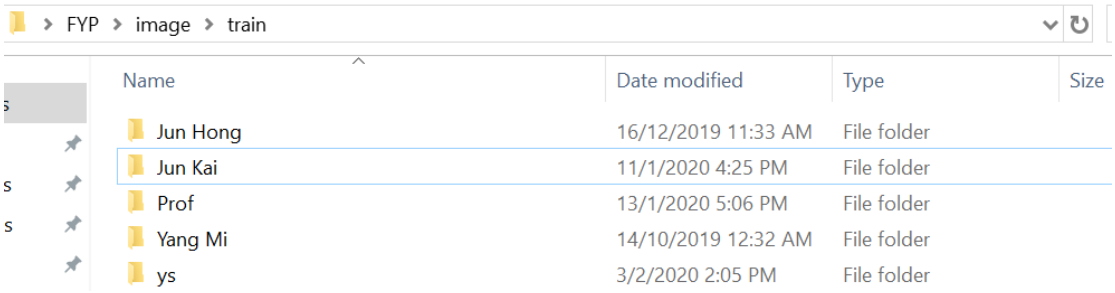

Figure 5. Successful Creation of Folder

Figure 6. Screenshot of 10 images

Chapter 4

Program Review (Train)

4.1 Face Recognition

The program review of the project in this section contains the backend of the program. From the detection of the face to the training of data using k-Nearest Neighbors algorithm and saving the model.

4.1.1 Face Detection

Face Detection is a common feature for camera devices. In the early 2000s, Paul Viola and Michael Jones invented an object detection framework (Viola-Jones) that was fast enough to run on cheap cameras. As it can be trained to detect a variety of object classes, Viola-Jones requires full view frontal upright faces and should not be tilted to either side (“Viola-Jones object detection framework,” February 2015). In 2005, Viola-Jones is used as an inspiration for the current method that used to detect faces.[5] The method used in this situation is called Histogram of Oriented Gradients, in short, HOG. The method is based on evaluating well-normalized local histograms of image gradient orientations in a dense grid.

The method converts the selected image into a monochrome image, and loops through

every single pixel on at a time. For every single looped pixel, the method compares the surrounding pixels and draw an arrow in the direction where the image is getting darker. After repeating the process for every single pixel in the image, the whole image will be replaced by an arrow. These arrows are called gradients and it shows the flow from light to dark across the image. The method solves the issue of having a different pixel value when presents with a dark and light image of the same person. In this situation, the method returns the same exact representation. It will then be further break up into small squares of 16x16 pixels where the method will average out the major direction of the gradients point. Next, the square in the image is replaced with the arrow directions that are the strongest. This results in the conversation of the original image into a simple representation that captures the basic structure of a face regardless of image brightness [6]. The program will find the face of the image that has a higher similarity to a known HOG pattern; which was extracted from a bunch of training faces.

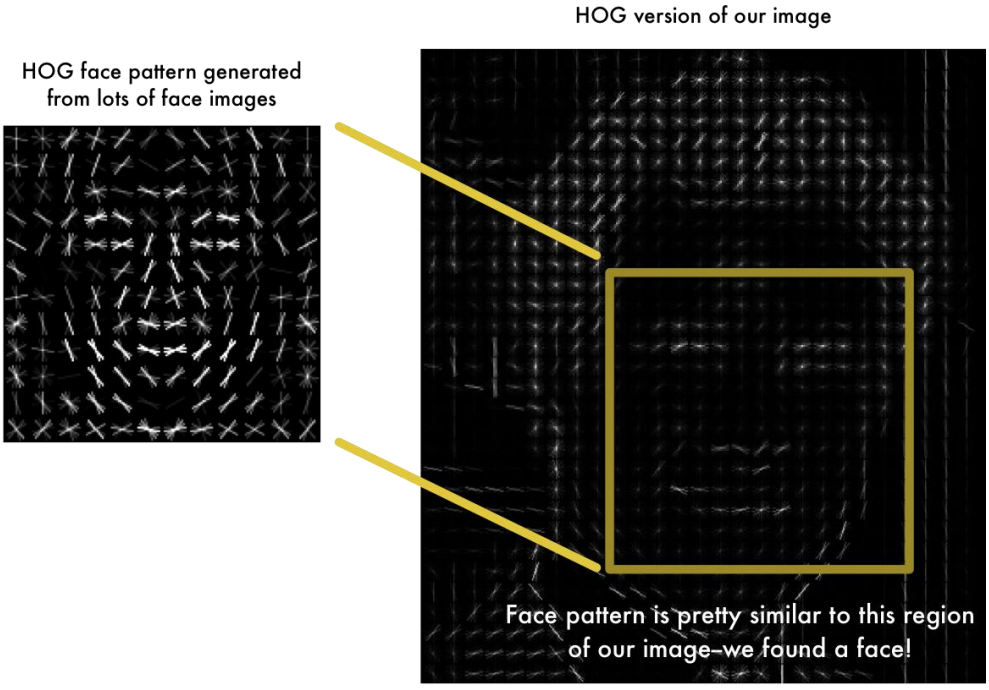

Figure 7. HOG presentation of the face pattern (Adam Geitgey 2016)

This technique is used by a built-in dlib class called get_frontal_face_detector(). It is used in the program to detect every face that appears in the stored image in the midst of looping every individuals' images that are stored in the training folder. The program will then bound the face portion with a bounding box and return a numeric value indicating how many faces are being detected in the image.

```
#Loop through each person in the training set
for class_dir in os.listdir(train_dir):
    if not os.path.isdir(os.path.join(train_dir, class_dir)):
        continue

    #Loop through each training image for the current person
    for img_path in image_files_in_folder(os.path.join(train_dir, class_dir)):
        image = face_recognition.load_image_file(img_path)
        face_bounding_boxes = face_recognition.face_locations(image)
```

Figure 8. Code for Face Detection using API: face_locations

4.1.2 Face Encoding

Face encoding trains a deep convolutional neural network to generate 128 measurements for each face. The training method employs a loss function called triplet loss. Triplet loss works by training on every three different sets; a training face image of a person called anchor, another image of the same known person called positive and an image of a totally different person called negative. The function calculates the Euclidean distance between the anchor and the positive image, and the anchor and the negative image. The training process aims to reduce the distance where similar images lie closer to each other while distinct images lie further in the embedding space. Eventually, the neural network learns to reliably generates 128 measurements for each person [7, 8].

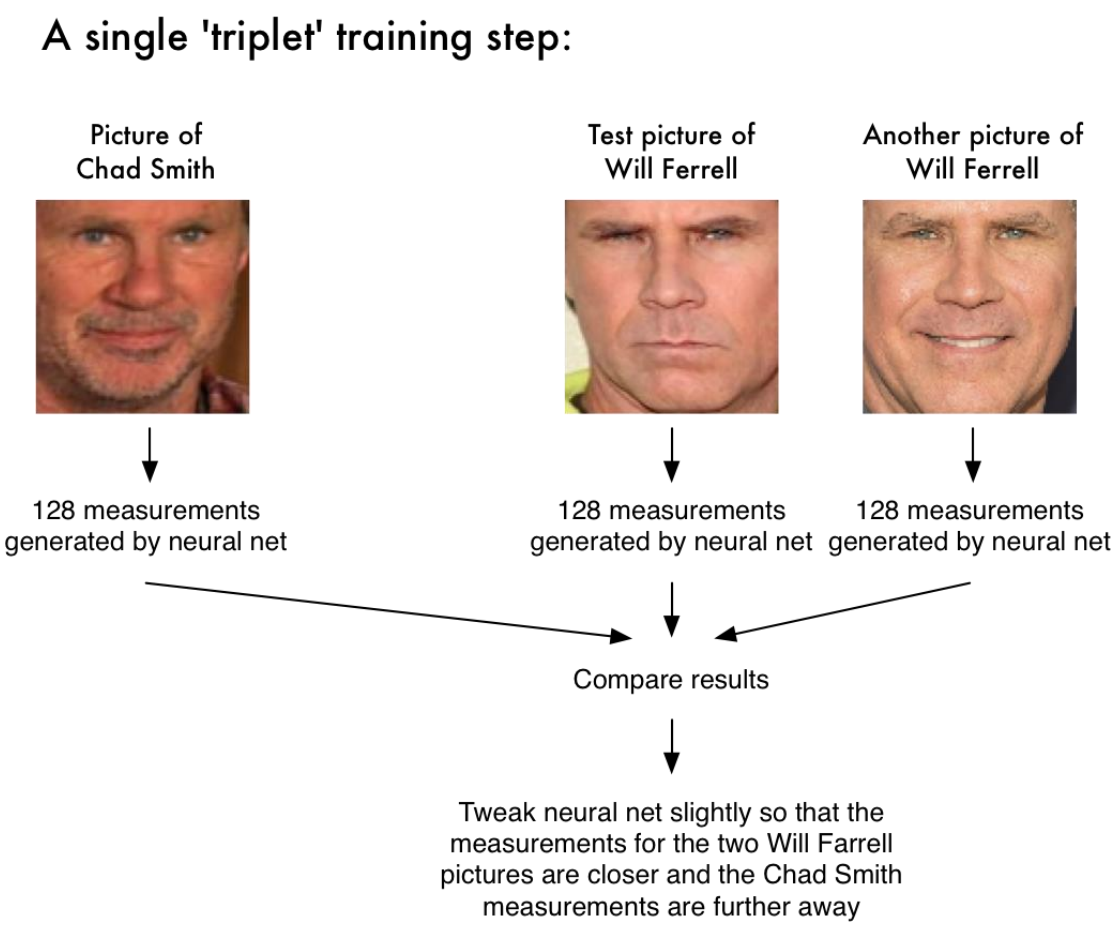

Figure 9 Triplet loss Step (Adam Geitgey 2016)

The program uses OpenFace, a free and open-source face recognition with deep neural networks for the generation of 128 measurements. The 128 facial measurements represent a generic face and the accuracy of OpenFace has been tested using a triplet loss function by FaceNet architecture of Google [9].

The function after ***Figure 7*** checks whether there is any face detected in the images. Before the program is going to train the classifier, it must not allow more than one or zero faces appearing in the image dataset, or else the result of the training will be affected and therefore, produce an inaccurate result. The program uses a dlib class: face_recognition_model_v1 and the chosen model is resnet_model_v1.

4.1.2.1 ResNet

Back in 2015, ResNet is a classic neural network used in many computer vision tasks. It was the most fundamental breakthrough in deep learning as ResNet allows users to train an extremely deep neural network with hundreds of layers successfully and still achieve persuasive performance [10] is the reason why it has been chosen. Due to the powerful representational ability of ResNet, the performance of face recognition has been boosted. In 2012, AlexNet that kick-started the focus on deep learning had only 8 convolutional layers, while the VGG network and GoogleNet had 19 and 22 layers respectively. However, increasing network depth does not work by simply stacking layers together. Deep networks are hard to train because of the notorious vanishing gradient problem. As the gradient is backpropagated to earlier layers, repeated multiplication may make the gradient extremely small. As a result, as the network goes deeper, the performance gets saturated or even starts degrading rapidly [11]. The solution to the problem is ResNet. It uses ***skip connection*** to add the output from an earlier layer to a later layer and this helps to mitigate the problem of vanishing gradient.

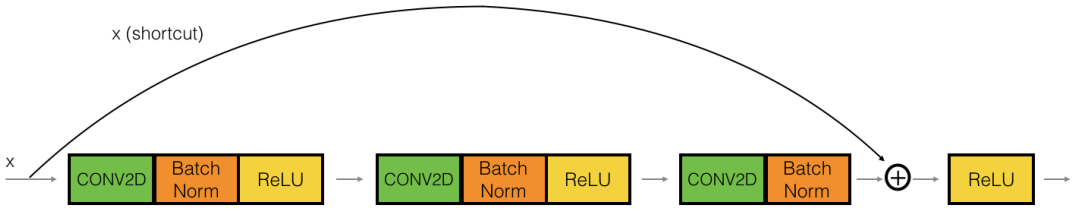

Figure 10. When x and x (shortcut) are the same shapes (Priya Dwivedi 2019)

As mentioned above, the used model is resnet_model_v1. The model is a ResNet network with 29 convolutional layers. It is created by Davis King in 2017 and released into the public domain. The model is a version of the ResNet-34 network from the paper Deep Residual Learning for Image Recognition by He, Zhang, Ren, and Sun with a few layers removed and the number of filters per layer reduced by half [10].

The network training started with randomly initialized weights and used a structured metric loss that tries to project all the identities into non-overlapping balls of radius 0.6. The loss is a type of pair-wise hinge loss that runs over all pairs in a mini-batch and includes hard-negative mining at the mini-batch level. The resulting model obtains a mean error of *0.993833* with a standard deviation of *0.00272732* on the LFW benchmark (Davis King, 2017).

4.1.2.2 Flow of the Program

To address the coding section of the program, the program appends the face encoding for the current image to the training set along with its class label and set up the k-NN classifier afterward.

```
if len(face_bounding_boxes) != 1:
    #if there are no people (or too many people) in a training image, skip the image
    if verbose:
        print("Image {} not suitable for training: {}".format(img_path, "Didn't find a face" if len(face_bounding_boxes) < 1
            else "Found more than one face"))
else:
    #Add face encoding for current image to the training set
    X.append(face_recognition.face_encodings(image, known_face_locations=face_bounding_boxes)[0])
    y.append(class_dir)
```

Figure 11 Validation check, encoding of detected face and comparing with images with a class label

4.1.3 K-Nearest Neighbor (k-NN)

The k-NN algorithm is the simplest algorithm for regression and classification in supervised learning. The advantage of using k-NN is that it quickly finds the closest object from the large training set. k-NN algorithm is also more suitable for recognizing a person based on their face images due to its small execution time and higher accuracy than other commonly used methods for example SVM, MLP or decision table [12]. The project uses Neighbors-based classification where it is a type of instance-based learning and non-generalizing learning. As it does not construct a general internal model, it simply stores instances of the training data [13]. Furthermore, it works by classifying an object based on the majority votes of its neighbors. Moreover, research has also shown that k-NN classification can be significantly improved by learning a distance metric from labeled examples [14], through the exploitation of the distance metric, the program is able to find the closest person from the training set. Therefore, the project uses Euclidean distance as the distance metric to measure the distance between images before making a prediction label on the face image.

4.1.3.1 Weighted k-NN

The project implements k-neighbors classification in KNeighborsClassifier function in Scikit-learn. As the optimal choice of the value of k is highly data-dependent, several issues are bound to affect the result [13]. One of the issues affecting the performance of k-NN algorithm will be the choice of the value of k *(neighbors that are used for weighting)*. If the value of k is too small, the algorithm will be more likely to be exposed to outliers and vice versa, the neighborhood may include too many points from other classes that affect the accuracy of the result. For that reason, the project square root the equation after the sum of the value of k according to the total number of detected faces. The purpose of doing this calculation is to automatically determine the number of neighbors to use for weighting in the k-NN classifier.

```
#Determine how many neighbors to use for weighting in the KNN classifier
if n_neighbors is None:
    n_neighbors = int(round(math.sqrt(len(X))))
    if verbose:
        print("Chose n_neighbors automatically: ", n_neighbors)
```

Figure 12. Calculation of k

4.1.3.2 K-D TREE

In the research proposed by Perez Martin, it is said that K-D tree might be the most popular partitioning trees for nearest neighbor search [15]. As K-D tree is a new type of data structure for associative searching, it recursively partitions the parameter space along the data axes and divides it into nested orthotropic regions into which data points are filed [16].

The limitation of the K-D tree is that it is unsuitable for finding nearest neighbors in high dimensions; namely Curse of Dimensionality. The Curse of Dimensionality causes every observation in the dataset to appear equidistant for all the data and especially for clustering methods that use distance metrics such as Euclidean distance to distinguish similarity between observation is severely affected. In a situation where all the distances are approximately equalivant to each other, all the observations will appear to be equally alike (as well as equally different), thus resulting in no meaningful clusters being formed [17]. Instead of using K-D tree, Ball Tree was proposed as the solution for this project.

4.1.3.3 Ball Tree

The program uses Ball Tree instead of K-D tree to address the inefficiencies of the K-D tree in higher dimensions because its performance degenerates significantly when the dimension increases [15].

```
def train(train_dir, model_save_path=None, n_neighbors=None, knn_algo='ball_tree', verbose=False):
    X = []
    y = []

    #Loop through each person in the training set
    for class_dir in os.listdir(train_dir):
        if not os.path.isdir(os.path.join(train_dir, class_dir)):
            continue
```

Figure 13. Screenshot of train's function using Ball tree

A Ball Tree is a special data structure that attempts to partition the training data in a way that only a portion of the training data has to be searched. The tree prune off paths that cannot contain points that are closer than the ones discovered and in order to achieve this, euclidean distance between two labeled data is needed. The Ball Tree data structure is efficient in situations where the number of dimensions is very large because the performance grows logarithmically and it will be efficient for the project in the future.

Furthermore, due to the spherical geometry of the ball tree nodes, it can outperform a K-D tree in high dimensions, however, the actual performance is highly dependent on the structure of the training data [13]. In this situation, the project is using a pre-trained model and the accuracy has reached 99.38% on the Labeled in the Wild which undoubtedly solves the data-dependent issue.

4.1.3.4 k-NN Classifier

Scikit-learn has two nearest neighbor classifiers, namely KNeighborsClassifier and RadiusNeighborsClassifier. These two are the most commonly used technique in k-neighbors classification. As mentioned above, KNeighborsClassifier implements learning based on the integer value of k whereas, RadiusNeighborsClassifier implements learning based on the number of neighbors within a fixed radius r of each training point. In addition, it is also used in a situation when the data is not uniformly sampled. However, for high dimensionality spaces, this method becomes less effective and more vulnerable to the Curse of Dimensionality. Therefore, this project uses KNeighborClassifier instead [13].

The parameters of the project where weights = ‘distance’ assigns the weights proportional to the inverse of the distance from the query point. In another word, it will tell the learner to weigh each vote from the observation by the distance from the image that the program is classifying.

```
#Create and train the KNN classifier
knn_clf = neighbors.KNeighborsClassifier(n_neighbors=n_neighbors, algorithm=knn_algo, weights='distance')
knn_clf.fit(X,y)
```

Figure 14. KNeighborsClassifier

4.1.3.5 Pickle Dump

The program uses Pickle for converting the result of k-classifier into a byte stream.

```
#Save the trained KNN classifier
if model_save_path is not None:
    with open(model_save_path, 'wb') as f:
        pickle.dump(knn_clf, f)

return knn_clf
```

Figure 15. saving the result of classifier into a byte stream file

Chapter 5

Program Review (Predict)

5.1 Predict

The program review of the project in this section contains a method for the prediction of the program. From passing in the k-classifier model into the predict function to recognizing the faces and returning of the class label. The distance threshold of the program is 0.4, meaning it will return a class label if the image has a 60% similarity. The used metric is Minkowski distance.

5.1.1 Condition Checking

The program validates the parameters that are passed into the function. The condition must be fulfilled for the program to do a prediction on the face. From **Section 4.1.3.5 Pickle Dump**, the program will unload the dump file that it has previously created.

```
def predict(frame, knn_clf=None, model_path=None, distance_threshold=0.4):
    #distance_threshold: the lower the number, the stricter the check is
    if knn_clf is None and model_path is None:
        raise Exception("Must supply knn classifier either through knn_clf or model_path")

    #Load a trained KNN model
    if knn_clf is None:
        with open(model_path, 'rb') as f:
            knn_clf = pickle.load(f)
```

Figure 16. Check condition

5.1.2 Flow of the Program

The program performs a face detection function. It is done so to detect the number of available faces that appears on the webcam. Afterward, the program will perform a conditional check. If there is no face appearing, it will return an empty list or else vice versa. Next, it will perform a face encoding on the detected face (*Refer to Section 4.1.2. Face Encoding*) in order to do the comparison between the detected face and the k-classifier model.

```
X_face_locations = face_recognition.face_locations(frame)

#If no faces are found in the image, return an empty result
if len(X_face_locations) == 0:
    return []

#Find encodings for faces in the test image
faces_encodings = face_recognition.face_encodings(frame, known_face_locations=X_face_locations)
```

Figure 17. Face Detection and Face Encoding

5.2 Computing the best matches

The program calculates the k-neighbors of a point and returns indices of and distances to the neighbors of each point. The calculation is done using the saved model: knn_clf. The selected neighbor is value 1 which means the program will select the first closest neighbor when calculating the distance. This portion of the code is to allow the program to take the first neighbor appearing in its distinct.

```
#Use the KNN to find the best mathes for the test face
closest_distances = knn_clf.kneighbors(faces_encodings, n_neighbors=1)
```

Figure 18. Distance computation

The program will compare the array values returned from *faces_encodings from Figure 16* and every array values of faces encoding in the saved model. The program only allows a threshold of less than 0.4 to be passed to the next step of the calculation. By sorting with the threshold, the program is able to distinguish the closest image folder to the detected face.

```
closest_distances: (array([[0.32587342]]), array([[5]], dtype=int64))
closest_distances: (array([[0.34787033]]), array([[2]], dtype=int64))
closest_distances: (array([[0.33990957]]), array([[5]], dtype=int64))
closest_distances: (array([[0.37524509]]), array([[2]], dtype=int64))
closest_distances: (array([[0.36544333]]), array([[2]], dtype=int64))
closest_distances: (array([[0.36676564]]), array([[2]], dtype=int64))
closest_distances: (array([[0.36267131]]), array([[2]], dtype=int64))
closest_distances: (array([[0.35636607]]), array([[2]], dtype=int64))
closest_distances: (array([[0.36963658]]), array([[2]], dtype=int64))
closest_distances: (array([[0.37042433]]), array([[2]], dtype=int64))
closest_distances: (array([[0.35933586]]), array([[2]], dtype=int64))
closest_distances: (array([[0.36182757]]), array([[2]], dtype=int64))
closest_distances: (array([[0.36665614]]), array([[2]], dtype=int64))
```

Figure 19. Calculation of closest distance

After the calculation of the closest distance, the program uses the detected face to match against it. This is to ensure that it can locate the class label with the closest distance. The result will be returned to the variable *are_matches.*

```
are_matches = [closest_distances[0][i][0] <= distance_threshold for i in range(len(X_face_locations))]
```

Figure 20. Matching the face with the distance

With the conditions are_matches and closest_distances, the program can effectively predict the class label for the detected face. The python zip function will match all the common features into a list. The class label is assigned to *pred*, location of the detected face is assigned to *loc* while the two conditions are assigned to *rec*. If the program is unable to find the result, it will return the word “unknown” with its respective location of the face. If all condition is true, then it will return the result of the respective class label and location of the face.

```
result = [(pred, loc) if rec else ("unknown", loc) for pred, loc, rec in zip(knn_clf.predict(faces_encodings), X_face_locations, are_matches)]
```

Figure 21. Selecting the class label

```
pred:  Jun Kai loc:  (79, 108, 120, 64)  rec:  True
pred:  Jun Kai loc:  (79, 108, 120, 64)  rec:  True
pred:  Jun Kai loc:  (78, 106, 120, 55)  rec:  True
pred:  Jun Kai loc:  (79, 103, 120, 59)  rec:  True
pred:  Jun Kai loc:  (79, 103, 120, 59)  rec:  True
pred:  Jun Kai loc:  (72, 112, 120, 60)  rec:  True
pred:  Jun Kai loc:  (72, 112, 120, 60)  rec:  True
pred:  Jun Kai loc:  (78, 118, 120, 66)  rec:  True
pred:  Jun Kai loc:  (74, 103, 118, 59)  rec:  True
pred:  Jun Kai loc:  (72, 101, 120, 49)  rec:  True
pred:  Jun Kai loc:  (72, 95, 120, 43)  rec:  True
pred:  Jun Kai loc:  (72, 95, 120, 43)  rec:  True
pred:  Jun Kai loc:  (72, 95, 120, 43)  rec:  True
pred:  Jun Kai loc:  (72, 95, 120, 43)  rec:  True
pred:  Jun Kai loc:  (72, 95, 120, 43)  rec:  True
pred:  Jun Kai loc:  (72, 95, 120, 43)  rec:  True
pred:  Jun Kai loc:  (67, 106, 118, 55)  rec:  True
pred:  Jun Kai loc:  (44, 106, 95, 55)  rec:  True
```

Figure 22. Printing the prediction

Chapter 6

System Design Overview

6.1 Flowchart of Rapid Facial Recognition System

This section illustrates the flowchart of the Rapid Facial Recognition System. It depicts how the program will proceed on recognizing acquaintances, unknown stranger and adding new acquaintances.

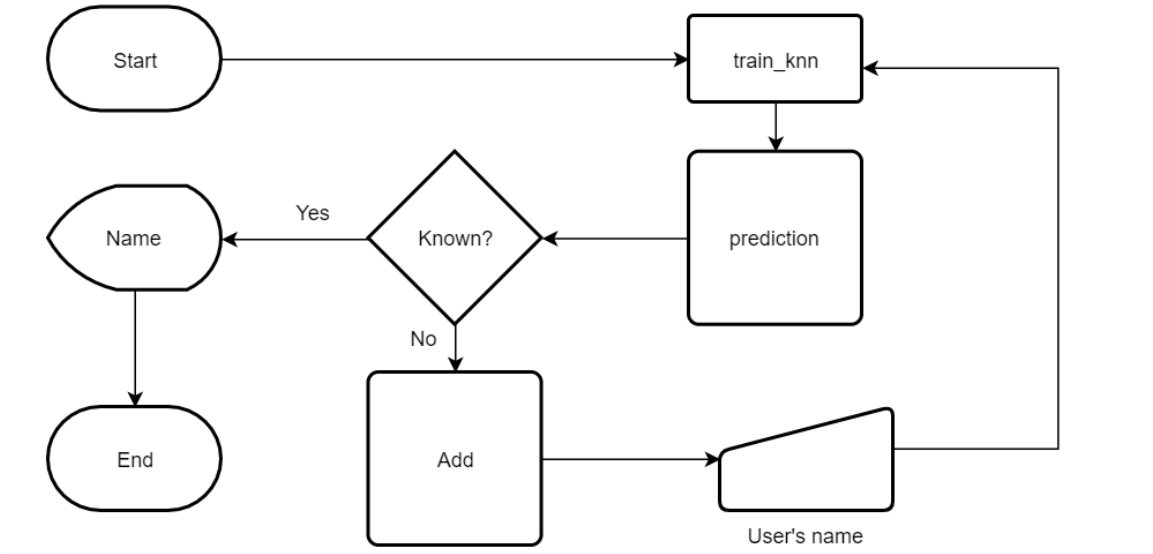

Figure 23. Flowchart of the Program

6.2 Graphical User Interface (GUI)

This section illustrates the options when the user is using the Rapid Facial Recognition System. In *Figure 21*, it illustrates the flow of adding an acquaintance into the program.

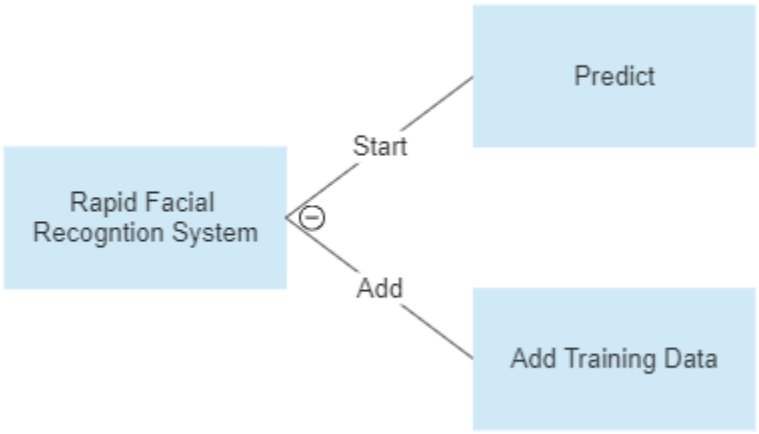

Figure 24. Program has only two options

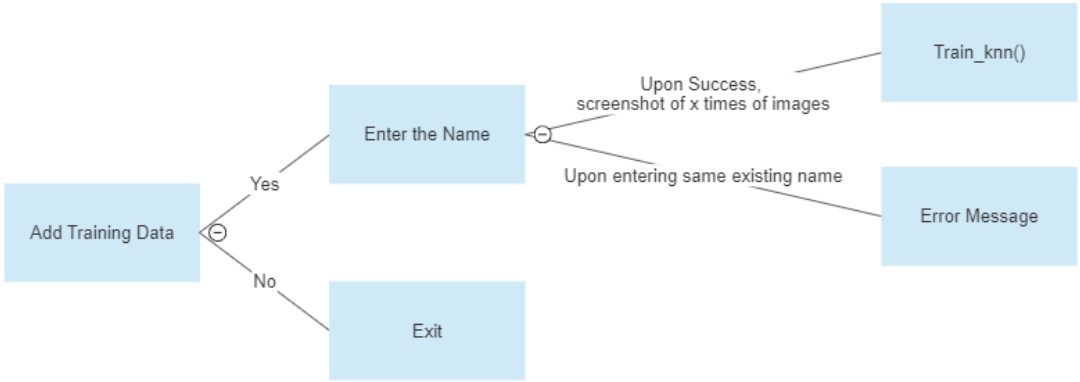

Figure 25. The decision tree on how "Add" is working

6.3 Train_knn()

This section illustrates the training flow of the k-classifier in the program.

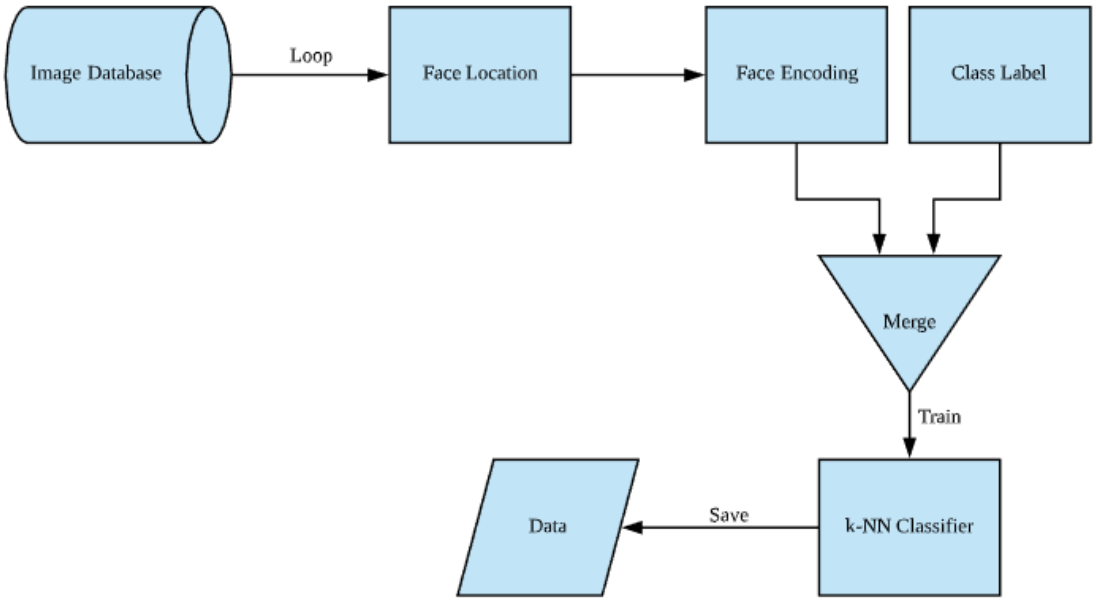

Figure 26. Training the classifier

6.4 Predict

The section illustrates the flow of the predict in the program after receiving the data (k-classifier model) from the train section. With the combination of model and real-time face detection, the program can achieve a rapid facial recognition.

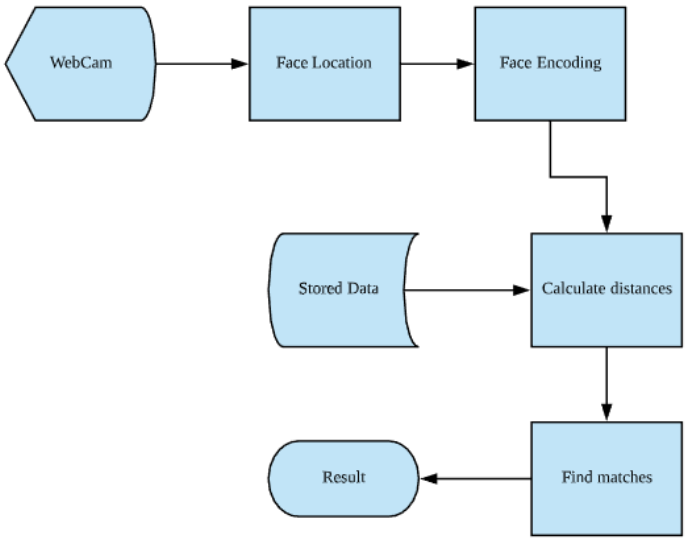

Figure 27. Predicting with model

6.4.1 Decision Tree of the Predicts Result

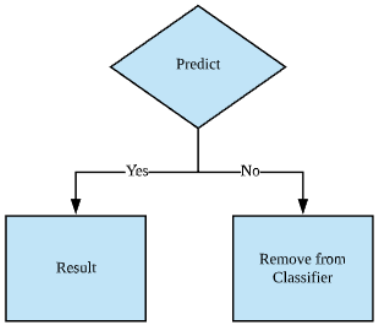

Figure 28. Predict Classes and Removing Classification aren't within the threshold

Chapter 7
Conclusions and Future Work

7.1 Conclusions

Face recognition techniques have been constantly improving throughout the years, and in order to fulfill the increasing demands on the recognition field, enhancement to areas such as techniques and methods had been in made. This greatly simplifies the recognition system and allows everyone to embed it into our daily life. For this project, the focus of it is on exploring a rapid face recognition system as a facial memory aid for users. As a result, the project has built a program that rapidly recognizes faces and displays the respective name of the face on the GUI. Furthermore, the program can also classify whether a person is an acquaintance, or an unknown stranger rapidly and it allows the addition of new acquaintance to the system.

7.2 Recommendation in Future Work

The project has a few places for improvement. First, the program has not tested the efficiency of the algorithm and techniques used. The current training data is small; therefore, the program can fulfill the condition, rapid recognition of individuals. However, if the data size increases exponentially, it is uncertain the program is still able to fulfill what it is intended to perform.

Second, the program can also have a better GUI to allow better user experience. This includes adding more interactive features, for example changing the number of times the camera takes a screenshot when adding the training data or adding to a pre-existing image folder.

Third, the location of the image stored will be an issue in the future. This is because the current size of the images is relatively small, however, the number of images will increase. The suggested solution for this problem will be having a database to store all the images.

Chapter 8
Reflection on Learning Outcome Attainment

This project consists of exploring face recognition and choosing the algorithm which allows the fastest recognition of an acquaintance. By completing this project, it has broadened my horizons on face recognition, for example; face detection, it taught me the history, algorithm, techniques, and method behind this simple detection. This project has also taught me the importance of communication between my FYP professor and me.

As there is a gap in the knowledge, often I have difficulty expressing my idea and what I intending to tell him. However, his advice on the progress on the project has greatly reduced many time-consuming portions and this helps me to advance further than I expected. As this project has many learning curves because it is a totally unknown field to me, time management becomes an issue because of the busy school schedule. I managed to learn the basics of the project and proceeded with the project smoothly. This project also helps me a lot in exploring the unknown ground of face recognition and machine learning and gives me an invaluable experience.

Chapter 9

References

1 Hassaballah, M., and Aly, S.: ‘Face Recognition: Challenges, Achievements, and Future Directions’, IET Computer Vision, 2015, 9, pp. 614-626

2 Masi, I., Tran, A.T., Hassner, T., Leksut, J.T., and Medioni, G.: ‘Do we really need to collect millions of faces for effective face recognition?’, in Editor (Ed.)^(Eds.): ‘Book Do we really need to collect millions of faces for effective face recognition?’ (Springer Verlag, 2016, edn.), pp. 579-596

3 G, Y.: ‘Which Python package manager should you use?’, towards data science, 2017

4 https://pysimplegui.readthedocs.io/en/latest/, accessed 29.02 2020

5 Geitgey, A.: ‘Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning’, in Editor (Ed.)^(Eds.): ‘Book Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning’ (Adam Geitgey, 2016, edn.), pp.

6 Dalal, N., and Triggs, B.: ‘Histograms of oriented gradients for human detection’, in Editor (Ed.)^(Eds.): ‘Book Histograms of oriented gradients for human detection’ (2005, edn.), pp. 886-893 vol. 881

7 Paul, J.: ‘The one with Face Recognition.’, towards data science, 2019

8 Schroff, F., Kalenichenko, D., and Philbin, J.: ‘FaceNet: A unified embedding for face recognition and clustering’, in Editor (Ed.)^(Eds.): ‘Book FaceNet: A unified embedding for face recognition and clustering’ (2015, edn.), pp. 815-823

9 Kumar, S.: ‘Face Recognition using OpenFace’, 2019

10 He, K., Zhang, X., Ren, S., and Sun, J.: ‘Deep Residual Learning for Image Recognition’ (2016. 2016)

11 Dwivedi, P.: ‘Understanding and Coding a ResNet in Keras’, towards data science, 2019

12 Sasirekha, K., and Thangavel, K.: ‘Optimization of K-nearest neighbor using particle swarm optimization for face recognition’, Neural Comput. Appl., 2019, 31, (11), pp. 7935-7944

13 Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., and Louppe, G.: ‘Scikit-learn: Machine Learning in Python’, Journal of Machine Learning Research, 2012, 12

14 Weinberger, K.Q., and Saul, L.K.: ‘Distance Metric Learning for Large Margin Nearest Neighbor Classification’, J. Mach. Learn. Res., 2009, 10, pp. 207–244

15 Pérez-Martín, J.: ‘Face Recognition at scale’ (2017. 2017)

16 Bentley, J.L.: ‘MULTIDIMENSIONAL BINARY SEARCH TREES USED FOR

ASSOCIATIVE SEARCHING', Commun. ACM, 1975, 18, (9), pp. 509-517
17 Yiu, T.: 'The Curse of Dimensionality', towards data science, 2019

Chapter 10

Appendix (optional)

10.1 Train_Predict

```
import math
from sklearn import neighbors
import os
import os.path
import pickle
from PIL import Image, ImageDraw
import cv2
import numpy as np
import sys
import face_recognition
from face_recognition.face_recognition_cli import image_files_in_folder
import PySimpleGUI as sg

ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}

def train(train_dir, model_save_path=None, n_neighbors=None, knn_algo='ball_tree', verbose=False):
    X = []
    y = []

    #Loop through each person in the training set
    for class_dir in os.listdir(train_dir):
        if not os.path.isdir(os.path.join(train_dir, class_dir)):
            continue

        #Loop through each training image for the current person
        for img_path in image_files_in_folder(os.path.join(train_dir, class_dir)):
            image = face_recognition.load_image_file(img_path)
            face_bounding_boxes = face_recognition.face_locations(image)
            #print("face_bounding_boxes: ", face_bounding_boxes)


            if len(face_bounding_boxes) != 1:
                #if there are no people (or too many people) in a training image, skip the image
                if verbose:
                    print("Image {} not suitable for training: {}".format(img_path, "Didn't find a face" if len(face_bounding_boxes) < 1
                        else "Found more than one face"))
            else:
                #Add face encoding for current image to the training set
                X.append(face_recognition.face_encodings(image, known_face_locations=face_bounding_boxes)[0])
                y.append(class_dir)
```

```
    #Determine how many neighbors to use for weighting in the KNN classifier
    if n_neighbors is None:
        n_neighbors = int(round(math.sqrt(len(X))))
        if verbose:
            print("Chose n_neighbors automatically: ", n_neighbors)


    #Create and train the KNN classifier
    knn_clf = neighbors.KNeighborsClassifier(n_neighbors=n_neighbors, algorithm=knn_algo, weights='distance')
    knn_clf.fit(X,y)

    #Save the trained KNN classifier
    if model_save_path is not None:
        with open(model_save_path, 'wb') as f:
            pickle.dump(knn_clf, f)

    return knn_clf


def predict(frame, knn_clf=None, model_path=None, distance_threshold=0.4):
    #distance_threshold: the lower the number, the stricter the check is
    if knn_clf is None and model_path is None:
        raise Exception("Must supply knn classifier either through knn_clf or model_path")

    #Load a trained KNN model
    if knn_clf is None:
        with open(model_path, 'rb') as f:
            knn_clf = pickle.load(f)

    X_face_locations = face_recognition.face_locations(frame)

    #If no faces are found in the image, return an empty result
    if len(X_face_locations) == 0:
        return []

    #Find encodings for faces in the test image
    faces_encodings = face_recognition.face_encodings(frame, known_face_locations=X_face_locations)
```

```
    #Use the KNN to find the best mathes for the test face
    closest_distances = knn_clf.kneighbors(faces_encodings, n_neighbors=1)


    are_matches = [closest_distances[0][i][0] <= distance_threshold for i in range(len(X_face_locations))]


    print("are_matches", [closest_distances[0][i][0] for i in range(len(X_face_locations))])

    #Predict classes and remove classification aren't within the threshold


    result = [(pred, loc) if rec else ("unknown", loc) for pred, loc, rec in zip(knn_clf.predict(faces_encodings), X_face_locations, are_matches)]

    #print ("result: ", result)
    return result

```

```
def train_knn():
    print("Training KNN classifier...")
    classifier = train("image/train", model_save_path="trained_knn_model.clf", n_neighbors = 2)
    print("Training complete!")
```

10.2 GUI

```python
import cv2, PySimpleGUI as sg
import os
import numpy as np
from webcam_train_predict import *

#train_knn()

layout = [[sg.Image(filename='')],
            [sg.Text('', justification='center', font='Helvetice 20', size=(10,1), key='output')],
            [sg.RButton('Add', size=(10,1), pad=((200, 0), 3), font='Any 14')]
            ]
#create the window and show it without the plot
window = sg.Window('Face Recognition', [[sg.Image(filename='', key='image')],], location=(800,400))

window.Layout(layout).Finalize()


cap = cv2.VideoCapture(0)       # Setup the camera as a capture device


while True:                     # The PSG "Event Loop"
    event, values = window.Read(timeout=20)      # get events for the window with 20ms max wait

    imgbytes = cv2.imencode('.png', cap.read()[1])[1].tobytes()
    window['image'].update(data=imgbytes)

    ret, frame = cap.read()

    #Resize to 1/4 size for faster face recognition processing
    small_frame = cv2.resize(frame, (0,0), fx=0.25, fy=0.25)
    #Convert the image from BGR to RGB
    rgb_small_frame = small_frame[:,:,::-1]
    #only process every other frame of video to save time
    # #Use trained classifier, make prediction for unknown images
    #Find all the people in the image using a trained classifier model
    predictions = predict(rgb_small_frame, model_path="trained_knn_model.clf")

    name = show_prediction_labels_on_webcam(frame, predictions)
```

```python
    if event is None:
        break
        # if user closed window, quit
    elif event is 'Add':
        user = sg.PopupGetText('Enter the name of the person', 'Add Training Data')

        if os.path.exists("image/train/"+user):
            sg.Popup('User with this '+user+' already exist.', 'Please try again with new name or press ctrl+c to quit')
            #print('User with this name already exist. Please try again with new name or press ctrl+c to quit')
            user = sg.PopupGetText('Enter the name of the person', 'Add Training Data')
        os.makedirs("image/train/"+user)
        count=0
        while count<10:
            ret, frame = cap.read()
            cv2.imwrite("image/train/"+user+"/"+str(count)+".jpg",frame)
            count=count+1
        train_knn()
    elif event is not None:
        # if len(predictions) > 1:
        #     name_list = []
        #     for name, (top, right, bottom, left) in predictions:
        #         font = cv2.FONT_HERSHEY_DUPLEX
        #         name_list.append(name)

        #     print(len(predictions), ": ", name_list)
        #     window['output'].update(name_list)
        # else:
        if name is not 'unknown':
            window['output'].update(name)
        else:
            name = 'unknown'
            window['output'].update(name)

```