**CS-5710 Machine Learning Project Final Increment**

MACHINE LEARNING TO IMPROVE THE EASE IN ATTENDANCE TAKING USING FACE RECOGNITION USING KNN

Submitted by T. JOHN EMMANUEL (700742724)  
Under the esteemed Guidance of Professor Muhammad Zubair Khan.

**Introduction:**

The recognition of the face is one of the most important and secure identifications of a human input formats for secure authentication. The idea of face recognition has been around for a long time, but the development of automated face recognition systems began in the 1960s and 1970s. One of the earliest pioneers in this field was Woodrow Bledsoe, who worked on facial recognition at the Panoramic Research Laboratory at Stanford University in the 1960s. Bledsoe's work involved manually marking key facial features on photographs and using a computer to measure the distances between these features for identification purposes. The initial ideas for face recognition were started around 1960s and 70s with the development of pattern recognition and visual abilities of computer architecture. The earliest full model was developed by Woodrow Bledsoe in the 1960s and it was called as “Candide” system. So, in this project I have decided to learn how Machine Learning can improve this recognition of human facial structures and Identify individuals accurately as it can be useful to multiple fields in the technologically advanced world.

**Objective:**

The main objective of this project is to develop a robust face recognition system that can accurately identify individuals in real-time using the stored face details. The algorithm uses the KNN to match the extracted features of the input face and match it with the one that has the most similar facial features in the database. The K-nearest neighbors (KNN) method is a face recognition technique that aims to create a system that can properly recognize and categorize people based on their facial attributes. Face recognition can be performed using KNN, a well-liked machine learning method for classification problems, by comparing the distances between feature vectors of known faces and an unknown face to identify the most similar faces. The important goal is to achieve high accuracy and in an efficient programming structure allowing for a domain to have a quick and reliable recognition of individuals.

**Motivation:**

Face recognition has numerous practical applications, including security systems, access control, surveillance, and personalized user experiences. The motivation behind this project is, by using this facial recognition one can gain easy access to their specified roles using their facial data rather than an input text which is not a reliable way to access a highly confidential data or role. The main motivation doing this using KNN is its simplicity, efficacy, and adaptability for classification tasks are the driving forces behind its use in face recognition. The main justifications would be KNN is a Straightforward and Natural Method, it’s Effective in High-Dimensional Spaces, Non-parametric Learning and its Resistance to Change.

**Evaluation:**

Metrics including accuracy, precision, recall, and F1 score can be used to assess the face recognition system's performance. The accuracy shows the proportion of tested faces that could be properly identified. Recall represents the percentage of correctly recognized faces out of all actual faces, whereas precision measures the proportion of correctly recognized faces out of all positive identifications. The F1 score offers a balanced evaluation metric by combining precision and recall into a single value.

**Significance:**

The K-nearest neighbors (KNN) technique for face recognition is very important in many disciplines. The importance of face recognition using KNN can include. Security and monitoring, Authentication via biometrics, A Customizable User Experience, Access Management Systems, Human-Computer Interaction, Forensic Examining**.** Systems for face recognition can automate access control procedures, and enable accurate identification of people, all of which can increase security. Face recognition can also improve individualized user experiences in programs like social media, online shopping, and entertainment. This project advances face recognition technology, opening the door for future systems that are more effective and precise.

**Execution:**

The project implementation involves several steps. At first, we import all the required libraries such as OpenCV, NumPy, Pickle and KN-Neighbor classifier from scikit library. The face data is then used to train a KNN classifier, where each face is represented by a feature vector and it accumulates the vectorized values for the comparison. During the recognition phase, the system captures video frames, detects faces using a Haar cascade classifier, extracts feature from the detected faces, and matches them against the features in the trained KNN model. The recognized individuals are displayed on the video capture box, along with bounding boxes around their faces and their names displayed. This identification can be used for several purposes and one of which is gathering large attendance in any domain in mere seconds which could save potential time and effort, thus leaving more time for important activities. The interpreter used is PyCharm as I wanted to store the modules in my personal computer rather than having them online if I went with Google Collab

Complete Execution in Detail:

CODE FILES:

ADD\_DATA.py :

import cv2  
import numpy as np  
import os  
import pickle  
  
def add\_data():  
 face\_data = []  
 i = 0  
 cam = cv2.VideoCapture(0)  
  
 facec = cv2.CascadeClassifier('data/haarcascade\_frontalface\_default.xml')  
  
 name = input('Enter your name --> ')  
 ret = True  
  
 while(ret):  
 ret, frame = cam.read()  
 if ret == True:  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 face\_coordinates = facec.detectMultiScale(gray, 1.3, 4)  
  
 for (x, y, w, h) in face\_coordinates:  
 faces = frame[y:y+h, x:x+w, :]  
 resized\_faces = cv2.resize(faces, (50, 50))  
  
 if i % 10 == 0 and len(face\_data) < 10:  
 face\_data.append(resized\_faces)  
 cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)  
 i += 1  
  
 cv2.imshow("Extracting Features (WILL EXIT AUTOMATICALLY) ", frame)  
  
 if cv2.waitKey(1) == 27 or len(face\_data) >= 10:  
 break  
 else:  
 print('error')  
 break  
  
  
 cv2.destroyAllWindows()  
 cam.release()  
 face\_data = np.asarray(face\_data)  
 face\_data = face\_data.reshape(10, -1)  
  
 if 'names.pkl' not in os.listdir('data/'):  
 names = [name]\*10  
 with open('data/names.pkl', 'wb') as f:  
 pickle.dump(names, f)  
 else:  
 with open('data/names.pkl', 'rb') as f:  
 names = pickle.load(f)  
  
 names = names + [name]\*10  
 with open('data/names.pkl', 'wb') as f:  
 pickle.dump(names, f)  
  
  
 if 'faces.pkl' not in os.listdir('data/'):  
 with open('data/faces.pkl', 'wb') as w:  
 pickle.dump(face\_data, w)  
 else:  
 with open('data/faces.pkl', 'rb') as w:  
 faces = pickle.load(w)  
  
 faces = np.append(faces, face\_data, axis=0)  
 with open('data/faces.pkl', 'wb') as w:  
 pickle.dump(faces, w)  
  
  
def face\_data():  
 with open('data/names.pkl', 'rb') as file:  
 data = pickle.load(file)  
 x = 1  
 while x < len(data):  
 print(data[x])  
 x = x + 10  
def avail():  
 with open('data/names.pkl', 'rb') as file:  
 data = pickle.load(file)

**Explanation for ADD\_DATA :**

The ADD\_DATA Python script performs face data collection and management for face recognition using the OpenCV library and pickle module. The script consists of two main functions: `add\_data()` and `face\_data()`, as well as a helper function `avail()`.So let’s analyze the code and summarize its functionality.

The `add\_data()` function is responsible for capturing face images from a webcam and storing them for training a face recognition model. It begins by initializing variables, including an empty list `face\_data` to store the face images. The script then uses the OpenCV `VideoCapture()` function to access the webcam. It loads the pre-trained Haar cascade classifier for face detection from the file `'data/haarcascade\_frontalface\_default.xml'`.

The user is prompted to enter their name through the console. The script enters a loop that continuously reads frames from the webcam. It converts each frame to grayscale and applies the face detection algorithm to detect faces in the image. For each detected face, it crops the region of interest (ROI) from the frame and resizes it to a fixed size of 50x50 pixels. If the number of collected face images (`i`) is a multiple of 10 and the number of stored face images (`face\_data`) is less than 10, it appends the resized face image to `face\_data`. Additionally, it draws a rectangle around the detected face on the frame.

The frame with the rectangle overlay is displayed in a window titled "Extracting Features (WILL EXIT AUTOMATICALLY)". The loop continues until either the user presses the Esc key or the required number of face images (10) is collected. Once the loop terminates, the OpenCV windows are closed, and the webcam is released.

The collected face images in `face\_data` are converted to a NumPy array, reshaped into a 10x2500 matrix (10 samples of 50x50 images), and stored in the file `'data/faces.pkl'`. The script checks if the file `'data/names.pkl'` exists. If it doesn't, it creates a new file and saves a list of 10 copies of the user's name in it. Otherwise, it appends 10 copies of the user's name to the existing list and updates the file.

The `face\_data()` function loads the names from the file `'data/names.pkl'` and prints them to the console. It iterates through the list of names, starting from index 1 and incrementing by 10, to print each name. This function allows users to view the names associated with the collected face images.

The `avail()` function loads the names from the file `'data/names.pkl'`. It does not perform any further operations but provides access to the loaded names for potential future use.

In a short form as a ML Student all I can conclude is that the provided code allows users to collect face images from a webcam, store them along with corresponding names, and retrieve the names associated with the collected face images. This data can be used for training a face recognition model using machine learning algorithms.

RECOG\_DATA.py:

import cv2  
import numpy as np  
import pickle  
from sklearn.neighbors import KNeighborsClassifier  
from ADD\_DATA import add\_data  
  
  
def recog\_data():  
 with open('data/faces.pkl', 'rb') as w:  
 faces = pickle.load(w)  
  
 with open('data/names.pkl', 'rb') as f:  
 labels = pickle.load(f)  
  
 facec = cv2.CascadeClassifier('data/haarcascade\_frontalface\_default.xml')  
  
 print('Shape of Faces matrix --> ', faces.shape)  
  
 knn = KNeighborsClassifier(n\_neighbors=5)  
 knn.fit(faces, labels)  
 cam = cv2.VideoCapture(0)  
  
 while True:  
 ret, fr = cam.read()  
 if ret == True:  
 gray = cv2.cvtColor(fr, cv2.COLOR\_BGR2GRAY)  
 face\_coordinates = facec.detectMultiScale(gray, 1.3, 5)  
  
 for (x, y, w, h) in face\_coordinates:  
 fc = fr[y:y + h, x:x + w, :]  
 r = cv2.resize(fc, (50, 50)).flatten().reshape(1, -1)  
 text = knn.predict(r)  
 cv2.putText(fr, text[0], (x, y - 15), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 0), 2)  
 cv2.rectangle(fr, (x, y), (x + w, y + w), (0, 0, 255), 2)  
  
 cv2.imshow('ESC TO EXIT', fr)  
 if cv2.waitKey(1) == 27:  
 break  
 else:  
 print("error")  
 break  
  
 cv2.destroyAllWindows()  
def main():  
 recog\_data()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Explanation for RECOG\_DATA:**

The RECOG\_DATA.py implements a face recognition system using the K-Nearest Neighbors (KNN) algorithm and OpenCV (Open Source Computer Vision Library). The system is capable of recognizing faces in real-time through a webcam feed.

The code begins by importing the required modules: cv2 for computer vision operations, numpy for array manipulation, pickle for serializing and deserializing Python objects, and KNeighborsClassifier from the scikit-learn library for implementing the KNN algorithm. It also imports the add\_data function from ADD\_DATA module, which presumably handles adding face data to the system.

The main function, recog\_data, is defined to perform face recognition. It loads the preprocessed face data and corresponding labels from the pickle files 'faces.pkl' and 'names.pkl', respectively. The face data is a numpy array of flattened face images, and the labels represent the names of the individuals associated with each face.

Next, the Haar cascade classifier, 'haarcascade\_frontalface\_default.xml', is loaded using the cv2.CascadeClassifier class. This classifier is used for face detection and identifies regions of interest (ROI) corresponding to faces within an input frame.

The KNN classifier is then instantiated with n\_neighbors set to 5. The faces and labels data are used to train the classifier using the fit() method.

The code initializes the webcam feed using cv2.VideoCapture(0). It enters a while loop where it continuously captures frames from the webcam. Each frame is converted to grayscale using cv2.cvtColor(). The face\_coordinates variable stores the detected face regions in the grayscale frame using the detectMultiScale() method.

The code then iterates over each detected face region and performs recognition. It extracts the face region from the original color frame, resizes it to a standard size of 50x50 pixels, and flattens it into a 1D array. This flattened array is then reshaped into a 2D array with a single sample.

The KNN classifier's predict() method is used to predict the label/name corresponding to the face. The recognized label is displayed on the frame using cv2.putText(), and a bounding box is drawn around the face region using cv2.rectangle().

The processed frame with annotations is displayed in a window using cv2.imshow(). The loop continues until the 'Esc' key is pressed, which breaks out of the loop and exits the application.

Finally, the code calls cv2.destroyAllWindows() to close all open windows and cleanup resources.

This in simple is a face recognition system using the KNN algorithm and OpenCV. It loads preprocessed face data, trains a KNN classifier, captures frames from a webcam, detects faces in the frames, and performs real-time face recognition by associating labels with the detected faces. You’ll see me performing the differential recognitions of two sample sets as me and one of my favorite and legendary ring fighter.

main.py:

import tkinter as tk  
from tkinter import messagebox  
from PIL import ImageTk, Image  
from ADD\_DATA import add\_data, face\_data  
from RECOG\_DATA import recog\_data  
import pickle  
import os.path  
  
window = tk.Tk()  
window.title("Command Center")  
window.geometry("900x750")  
image\_path = "data/face\_recognition\_image.png"  
image = Image.open(image\_path)  
image = image.resize((800,500), Image.ANTIALIAS)  
photo = ImageTk.PhotoImage(image)  
image\_label = tk.Label(window, image=photo)  
image\_label.pack(pady=20)  
  
button\_frame = tk.Frame(window)  
button\_frame.pack()  
  
def total\_names():  
 face\_data()  
 messagebox.showinfo('Sucessfull','Check in the Console')  
def add():  
 messagebox.showinfo('Attention!!', 'Enter your name in the console')  
 add\_data()  
def iftru():  
 path = 'C:/Users/John/Desktop/ML FINAL PROJECT/Final Project\_Face-Recognition-using-KNN-main/Face-Recognition-using-KNN-main/data/names.pkl'  
 check\_file = os.path.exists(path)  
 if check\_file == False:  
 messagebox.showinfo('ERROR','NO INFORMATION IN THE DATABASE TO START TO BEGIN, ADD NEW FACE')  
 else:  
 recog\_data()  
  
  
animate\_button = tk.Button(button\_frame, text="START RECOGNIZING", command=iftru, width=25,fg="white", bg="green")  
animate\_button.grid(row=0, column=1, padx=20, pady=10)  
  
click\_button = tk.Button(button\_frame, text="ADD NEW FACE DATA", command=add, width=25,fg="white", bg="green")  
click\_button.grid(row=2, column=1, padx=10, pady=10)  
  
click\_button = tk.Button(button\_frame, text="NAMES IN DATABASE", command=total\_names, width=25,fg="white", bg="green")  
click\_button.grid(row=3, column=1, padx=10, pady=10)  
  
  
click\_button = tk.Button(button\_frame, text="EXIT THE COMMAND CENTER", command=exit, width=25,fg="white", bg="red")  
click\_button.grid(row=4, column=1, padx=10, pady=10)  
  
  
  
window.mainloop()

**Explanation for main.py(My main Function):**

Having all the messed up coding lines in the other two python script files, I’ve decided that I had enough time to develop a GUI in order to make my Interaction with code clean. So in here, the code is about graphical user interface (GUI) application implemented using the tkinter library in Python. The application serves as a command center for face recognition using the K-nearest neighbors (KNN) algorithm and the OpenCV library.

Upon execution, a window titled "Command Center" is created with dimensions of 900x750 pixels. The GUI contains an image label that displays a face recognition image. The image is loaded using the PIL (Python Imaging Library) module, specifically the ImageTk and Image classes.

The GUI also includes a button frame where several buttons are placed for different functionalities. The buttons are defined using the tk.Button class and are assigned specific commands to execute when clicked.

Here are my four main buttons in the GUI:

1. "START RECOGNIZING" Button:

- When clicked, it calls the "iftru" function.

- This function checks if the necessary data files for face recognition exist.

- If the files are present, it calls the "recog\_data" function, which initiates the face recognition process using the stored face data.

- If the files are not found, it displays an error message using the messagebox module.

2. "ADD NEW FACE DATA" Button:

- When clicked, it calls the "add" function.

- The function displays an informational message box instructing the user to enter their name in the console.

- It then calls the "add\_data" function, which uses the webcam to capture facial data and store it for future recognition.

3. "NAMES IN DATABASE" Button:

- When clicked, it calls the "total\_names" function.

- The function checks if the file containing names in the database exists.

- If the file exists, it displays a success message using the messagebox module and calls the "face\_data" function to print the names in the console.

- If the file does not exist, it displays an error message using the messagebox module.

4. "EXIT THE COMMAND CENTER" Button:

- When clicked, it exits the application.

The code utilizes the pickle module to read and write data files in binary format. It uses the os.path module to check the existence of specific files in the file system.

So in final I had successfully set up a user-friendly GUI for a face recognition naming it command center. It allows users to perform actions such as starting face recognition, adding new face data, checking the names in the database, and exiting the application. The GUI provides visual feedback using images and message boxes, while the underlying functions handle the data processing and interaction with the user.

**References/Bibliography:**

1. Bledsoe, W. (1996). “Face Recognition and Identification”. Technical Report, Panoramic Research, Inc.

2. Belhumeur, P. N., Hespanha, J. P., & Kriegman, D. J. (1997). "Eigenfaces vs. Fisherfaces: Recognition using class specific linear projection." IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(7), 711-720.

3. Face Recognition using OpenCV and Python: https://www.datacamp.com/community/tutorials/face-recognition-python-opencv

4. Hastie, T., Tibshirani, R., & Friedman, J. (2009). "The Elements of Statistical Learning: Data Mining, Inference, and Prediction." Springer.

5. Bishop, C. M. (2006). "Pattern Recognition and Machine Learning." Springer.

6. Haar Cascade Classifiers for Object Detection: <https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html>

7. Pattern Recognition and Machine Learning by Christopher M. Bishop

8. K-Nearest Neighbors Algorithm Explained: <https://towardsdatascience.com/knn-algorithm-and-implementation-from-scratch-b9f9b739c02e>

9. Hands-On Machine Learning with Scikit-Learn and TensorFlow by Aurélien Géron

10. scikit-learn: Machine Learning in Python: <https://scikit-learn.org/>

11. Tkinter GUI Application Development Blueprints by Bhaskar Chaudhary

12. The official GitHub repository of PILLOW provides access to the source code, issue tracker, and examples that can help you understand how the library works. <https://github.com/python-pillow/Pillowhttps://github.com/python-pillow/Pillow>

13. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" by Aurélien Géron”

14. Deep Face Recognition" paper by Yaniv Taigman et al.: <https://www.cs.toronto.edu/~ranzato/publications/taigman_cvpr14.pdf>

15. FaceNet: A Unified Embedding for Face Recognition and Clustering" paper by Florian Schroff et al.: <https://arxiv.org/abs/1503.03832>