

SMART ATTENDANCE USING FACE RECOGNITION

I. INTRODUCTION :

Face Detection use a face detection algorithm to locate and extract faces from images or video frames. Feature Extraction: Extract features from the detected faces, such as facial landmarks or deep features obtained from convolutional neural networks (CNNs). Training: Train a machine learning model (e.g., SVM, k-NN, or CNN) using the extracted features and corresponding labels (e.g., names or IDs of individuals). Recognition: Given a new face, use the trained model to predict its identity based on the extracted features.

II. ABSTRACT :

The Smart Attendance Project aims to automate attendance tracking processes using advanced technologies such as RFID, biometrics, or facial recognition. By replacing traditional manual methods, it streamlines attendance management, enhances accuracy, and reduces administrative burden in educational institutions or workplaces. This abstract outlines the project's objectives, methodologies, and potential benefits for efficient attendance monitoring

III. METHOD DESCRIPTION :

In today's technological landscape, the automation of routine tasks is becoming increasingly prevalent, enhancing efficiency and accuracy across various domains. This paper presents the development of a smart attendance system leveraging machine learning techniques implemented in Python. The system aims to streamline traditional attendance tracking methods by automating the process through facial recognition technology.

The system consists of several key components, including data collection, preprocessing, feature extraction, training, and integration. Data collection involves gathering a dataset of facial images representing individuals whose attendance will be monitored. Preprocessing techniques such as face detection and alignment ensure uniformity and quality of the dataset. Feature extraction is performed using deep learning models to extract discriminative features from facial images.

Training of the machine learning model involves the utilization of algorithms such as Support Vector Machines (SVM) or Convolutional Neural Networks (CNN) to learn patterns and associations between facial features and individual identities. The trained model is then integrated into a user-friendly interface, allowing for the seamless capture of facial images and real-time attendance tracking.

The proposed smart attendance system offers several advantages over traditional methods, including increased accuracy, efficiency, and scalability. By automating the attendance tracking process, organizations can optimize resource allocation and minimize administrative overhead. Additionally, the system can be easily customized and deployed across various environments, making it adaptable to diverse use cases and settings.

Overall, the development of a smart attendance system using machine learning represents a significant advancement in attendance management practices, offering a robust and intelligent solution to address the evolving needs of modern organizations.

Software required :

Visual Studio Code /Spyder/Pycharm

Programming Language :

Python :

Python, with its rich ecosystem of libraries and frameworks, has become a popular choice for implementing face recognition systems due to its simplicity and versatility. Several libraries and tools are available in Python for face recognition, including OpenCV, Dlib, and Face Recognition.

Required Packages and installations :

Opencv ,Numpy , Pandas , Scikit learn, Csv, Pickles, Os, Win32, Streamlit ..,etc.

OpenCV (Open Source Computer Vision Library) is a popular open-source computer vision and machine learning software library. It provides various tools and functions for image and video analysis, including object detection, face recognition, image processing, and more. OpenCV is widely used in both academia and industry for applications such as robotics, augmented reality, surveillance, and medical imaging

Code for install :

```
pip install opencv-python
```

NumPy is a fundamental package for scientific computing with Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. NumPy is widely used in fields such as data science, machine learning, signal processing, and scientific computing.

Code for install :

```
pip install numpy
```

Scikit-learn is a popular machine learning library for Python. It provides a simple and efficient tool for data mining and data analysis, built on top of other Python libraries like NumPy, SciPy, and matplotlib. Scikit-learn includes a wide range of machine learning algorithms for classification, regression, clustering, dimensionality reduction, model selection, and preprocessing.

Code for install :

```
pip install scikit-learn
```

Pandas is a powerful and flexible open-source data analysis and manipulation library for Python. It provides easy-to-use data structures, such as Series (one-dimensional labeled arrays) and DataFrame (two-dimensional labeled data structures with columns of potentially different types), which are particularly useful for working with structured data like tabular data or time series.

Code for install :

```
pip install pandas
```

Streamlit is an open-source Python library that makes it easy to create web applications for machine learning, data science, and analytics. With Streamlit, you can quickly build interactive and customizable web-based interfaces using only Python code, without needing to write HTML, CSS, or JavaScript

Code for install :

```
pip install streamlit
```

win32 typically refers to the set of extensions, modules, and tools provided by the Python for Windows Extensions project. These extensions allow Python programs to interact with the Windows operating system, accessing functionalities not directly available through the standard Python library.

Code for install :

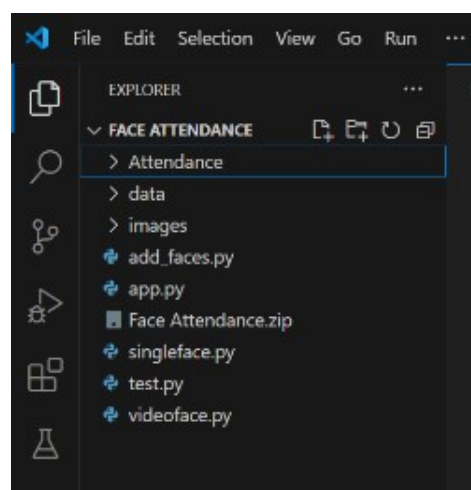
```
pip install pywin32
```

Steps following for the project :

In Python, implementing face recognition typically involves the following steps:

1. **Face Detection:** Use a face detection algorithm to locate and extract faces from images or video frames.
2. **Face Alignment** (Optional): Align faces to a standardized position to improve recognition accuracy. This step is particularly useful when dealing with faces in varying poses or orientations.
3. **Feature Extraction:** Extract features from the detected faces, such as facial landmarks or deep features obtained from convolutional neural networks (CNNs).
4. **Training:** Train a machine learning model (e.g., SVM, k-NN, or CNN) using the extracted features and corresponding labels (e.g., names or IDs of individuals).
5. **Recognition:** Given a new face, use the trained model to predict its identity based on the extracted features.
6. Face recognition using Python enables developers to build sophisticated applications for various purposes, including security, surveillance, biometrics, and personalization.

Folder Arrangements :



IV. SOURCE CODE :

Code for Add Faces as samples :

```
import cv2

import numpy as np

import os

import pickle

Data=cv2.CascadeClassifier('data/haarcascade_frontalface_default.xml')

video=cv2.VideoCapture(0)

faces_data=[]

i=0

name=input("Enter your name :")

while True:

    ret,frame=video.read()

    if ret==True:

        gray_image=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)

        faces=Data.detectMultiScale(gray_image,1.3,5)

        for x,y,w,h in faces:

            crop_img=frame[y:y+h,x:x+w,:]

            resized_img=cv2.resize(crop_img,(50,50))

            if len(faces_data)<=100 and i%10==0:

                faces_data.append(resized_img)

            i=i+1

            cv2.putText(frame,str(len(faces_data)),(50,50), cv2.FONT_HERSHEY_COMPLEX,

2, (50,50,255))

            cv2.rectangle(frame,(x,y),(x+w,y+h),(50,50,255),2)

        cv2.imshow('Frame',frame)

        key=cv2.waitKey(1)

        if (key==ord('q')) or len(faces_data)==100:
```

```
        break
video.release()
cv2.destroyAllWindows()
faces_data=np.asarray(faces_data)
faces_data=faces_data.reshape(100,-1)

if 'names.pkl' not in os.listdir('data/'):
    names=[name]*100
    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]*100
    with open('data/names.pkl','wb') as f:
        pickle.dump(names,f)

if 'face_data.pkl' not in os.listdir('data/'):
    with open('data/face_data.pkl','wb') as f:
        pickle.dump(faces_data,f)
else:
    with open('data/face_data.pkl','rb') as f:
        faces=pickle.load(f)
    faces=np.append(faces,faces_data,axis=0)
    with open('data/face_data.pkl','wb') as f:
        pickle.dump(faces,f)
```

Explanation :

This Python code is for capturing facial images using a webcam and storing them for face recognition purposes. Here's an explanation of the code:

1. Importing Libraries: The code starts by importing necessary libraries, including OpenCV (`cv2`), NumPy (`numpy`), operating system (`os`), and pickle for serialization.

2. Loading Haar Cascade Classifier: The Haar Cascade classifier (`haarcascade_frontalface_default.xml`) is loaded. This pre-trained classifier is used for face detection.

3. Initializing Variables: Variables like `video` for capturing video from the webcam, `faces_data` for storing captured facial images, and `i` for counting frames are initialized.

4. User Input: The user is prompted to enter their name, which will be associated with the captured facial images.

5. Video Capture Loop: The code enters a loop to continuously capture frames from the webcam. Within the loop:

- The `video.read()` function reads a frame from the webcam.
- The frame is converted to grayscale using `cv2.cvtColor()`.
- The `detectMultiScale()` function of the Haar Cascade classifier detects faces in the grayscale frame.
- For each detected face:
 - The face region is cropped.
 - The cropped image is resized to 50x50 pixels.
- If less than 100 images have been captured and every 10th frame is reached, the resized image is appended to `faces_data`.
- The count of captured images is displayed on the frame.
- A rectangle is drawn around the detected face on the frame.
- The frame with drawn rectangles is displayed.

6. Key Press Handling: The code checks if the 'q' key is pressed or if 100 images have been captured. If so, it breaks out of the loop.

7. Data Processing and Saving: After capturing images, the captured facial images are converted to NumPy array format, reshaped, and stored in a file named `face_data.pkl`. The associated names are stored in another file named `names.pkl`. If these files do not exist, they are created, and if they exist, new data is appended to them.

This code essentially creates a dataset of facial images labeled with the user's name for face recognition model training. Each time it's run, it captures additional images and appends them to the existing dataset.

Code to create and run Machine learning model :

```
from sklearn.neighbors import KNeighborsClassifier
```

```
import cv2
```

```
import numpy as np
```

```
import os
```

```
import pickle
```

```
import csv
```

```
import time
```

```
from datetime import datetime
```

```
from win32com.client import Dispatch
```

```
def speak(str1):
```

```
    speak=Dispatch(("SAPI.spVoice"))
```

```
    speak.speak(str1)
```

```
Data=cv2.CascadeClassifier('data/haarcascade_frontalface_default.xml')
```

```
video=cv2.VideoCapture(0)
```



```

with open('data/names.pkl','rb') as w:
    LABELS=pickle.load(w)
with open('data/face_data.pkl','rb') as f:
    FACES=pickle.load(f)

print("Shape of face matrix :",FACES.shape)

COL_NAMES=['NAMES','TIME']

model=KNeighborsClassifier(n_neighbors=5)
model.fit(FACES,LABELS)
while True:
    ret,frame=video.read()
    if ret==True:
        gray_image=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
        faces=Data.detectMultiScale(gray_image,1.3,5)
        for x,y,w,h in faces:
            crop_img=frame[y:y+h,x:x+w,:]
            resized_img=cv2.resize(crop_img,(50,50)).flatten().reshape(1,-1)
            prediction=model.predict(resized_img)
            ts=time.time()
            date=datetime.fromtimestamp(ts).strftime("%d-%m-%y")
            timestamp=datetime.fromtimestamp(ts).strftime("%H:&M:%S")
            exist=os.path.isfile("Attendance/Attendance_"+date+".csv")
            cv2.putText(frame,(str(prediction[0])),(x,y,15),cv2.FONT_HERSHEY_COMPLEX,1,(255,255,255),2)
            cv2.rectangle(frame,(x,y),(x+w,y+h),(50,50,255),1)
            attendance=[str(prediction[0]),str(timestamp)]
            cv2.imshow('Frame',frame)
            key=cv2.waitKey(1)
            if key==ord('o'):

```

```

speak("Attendance Taken")
time.sleep(2)
if exist:
    with open("Attendance/Attendance_"+date+".csv","a") as csvFile:
        writer=csv.writer(csvFile)
        writer.writerow(attendance)
    csvFile.close()
else:
    with open("Attendance/Attendance_"+date+".csv","a") as csvFile:
        writer=csv.writer(csvFile)
        writer.writerow(COL_NAMES)
        writer.writerow(attendance)
    csvFile.close()
if (key==65 or key==97):
    break
video.release()
cv2.destroyAllWindows()

```

Explanation :

This Python script is designed to perform real-time face recognition using a pre-trained K-Nearest Neighbors (KNN) classifier. Let's break down the code:

1. Imports: The script imports necessary libraries, including `KNeighborsClassifier` from `sklearn.neighbors`, `cv2` for OpenCV, `numpy` for numerical computations, `os` for operating system-related tasks, `pickle` for serialization, `csv` for handling CSV files, `time` for time-related operations, `datetime` for working with dates and times, and `Dispatch` from `win32com.client` for text-to-speech functionality.

2. Function Definition: The `speak()` function uses Windows Speech API (SAPI) to enable text-to-speech functionality.

3. Loading Pre-trained Data: The script loads pre-trained data (names and facial images) from the `names.pkl` and `face_data.pkl` files, respectively.

4. Initializing Model: It initializes a KNN classifier with 5 neighbors and fits it to the loaded facial data.

5. Video Capture Loop: The script enters a loop to continuously capture frames from the webcam. Within the loop:

- It reads a frame from the webcam.
- Converts the frame to grayscale and detects faces using the Haar Cascade classifier.
- For each detected face:
 - Crops and resizes the face image to 50x50 pixels.
 - Flattens and reshapes the image for prediction.
 - Uses the trained model to predict the label (name) of the face.
 - Records the current timestamp.
 - Draws a rectangle around the detected face and displays the predicted label.
- If the 'o' key is pressed, it announces the attendance taken, writes the attendance data to a CSV file named according to the current date, and appends or creates the CSV file accordingly.
- If the 'a' key (either uppercase or lowercase) is pressed, it breaks out of the loop, ending the program.

6. Release Resources: After the loop exits, it releases the video capture resources and closes all OpenCV windows.

This script essentially performs real-time face recognition, displaying the recognized name on the video feed and allowing the user to record attendance by pressing a key.

Run the Attendance on the browser using streamlit :

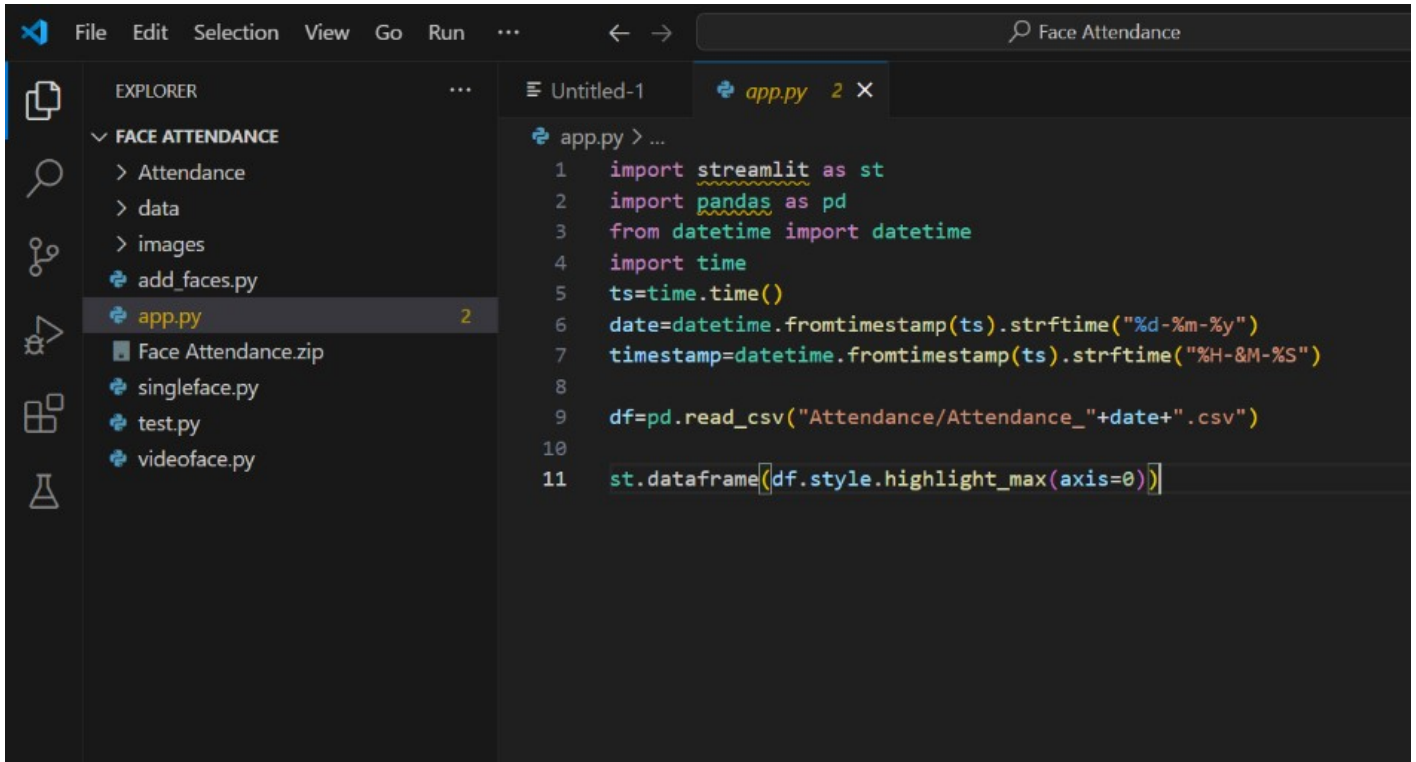
```
import streamlit as st
import pandas as pd
from datetime import datetime
import time
ts=time.time()
date=datetime.fromtimestamp(ts).strftime("%d-%m-%y")
timestamp=datetime.fromtimestamp(ts).strftime("%H-&M-%S")
df=pd.read_csv("Attendance/Attendance_"+date+".csv")
st.dataframe(df.style.highlight_max(axis=0))
```

Explanation :

1. **Imports:** The script imports necessary libraries, including **streamlit** for creating web applications, **pandas** for data manipulation, and **datetime** for working with dates and times.
2. **Timestamp Generation:** It generates the current date and timestamp using **time.time()** and **datetime.fromtimestamp()**. This is used to construct the filename for the CSV file containing attendance data.
3. **Reading CSV File:** The script reads the CSV file containing attendance data for the current date using **pd.read_csv()**. It assumes that the CSV file is located in the "Attendance" directory and follows the naming convention "Attendance_DD-MM-YY.csv", where "DD-MM-YY" represents the current date.
4. **Displaying Data:** The attendance data read from the CSV file is displayed in a DataFrame using **st.dataframe()**. The **.style.highlight_max(axis=0)** method is used to highlight the maximum value in each column.
5. **Streamlit Web Application:** When executed, Streamlit automatically converts the script into a web application, which can be accessed through a web browser. The attendance data is displayed in a tabular format within the web application.

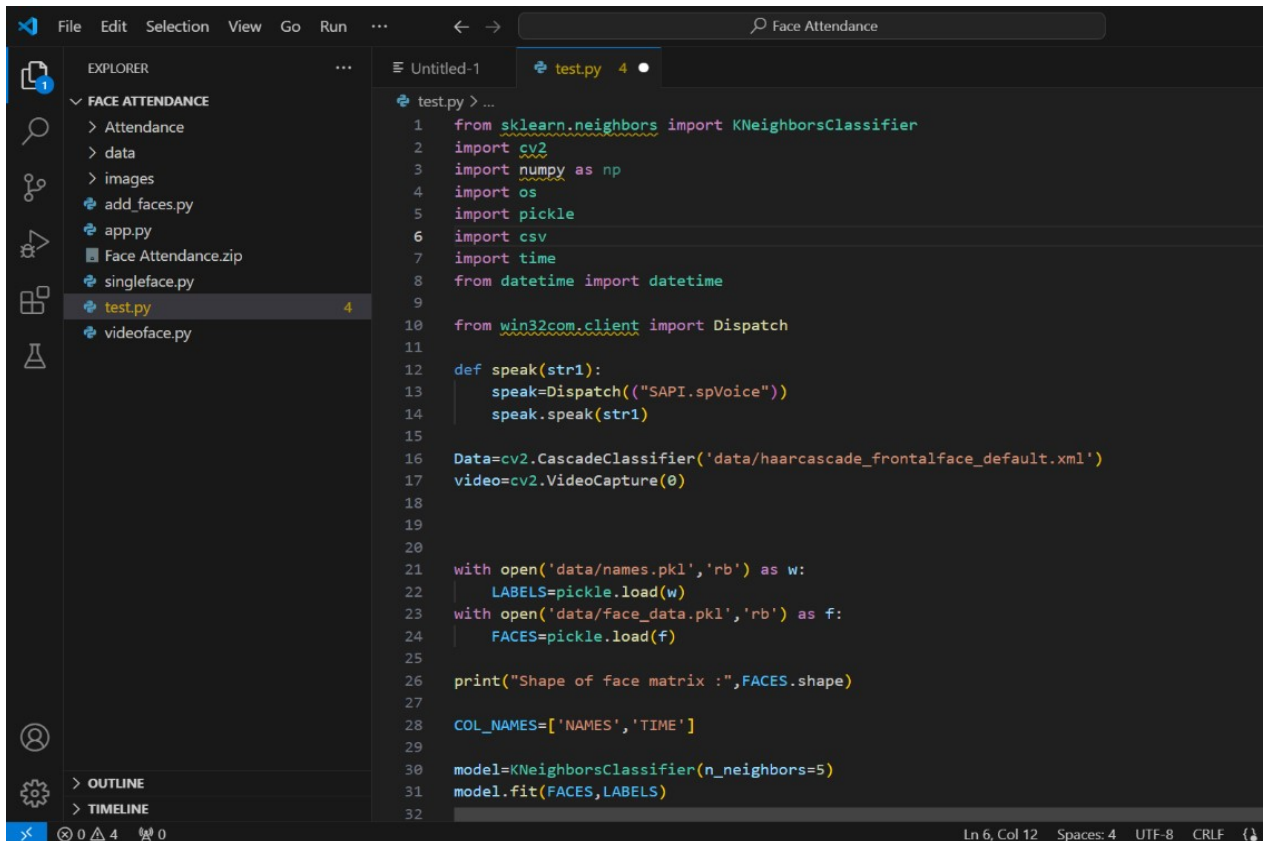
Code Screenshots :

app.py is used for stream our attendance details in browser using streamlit.



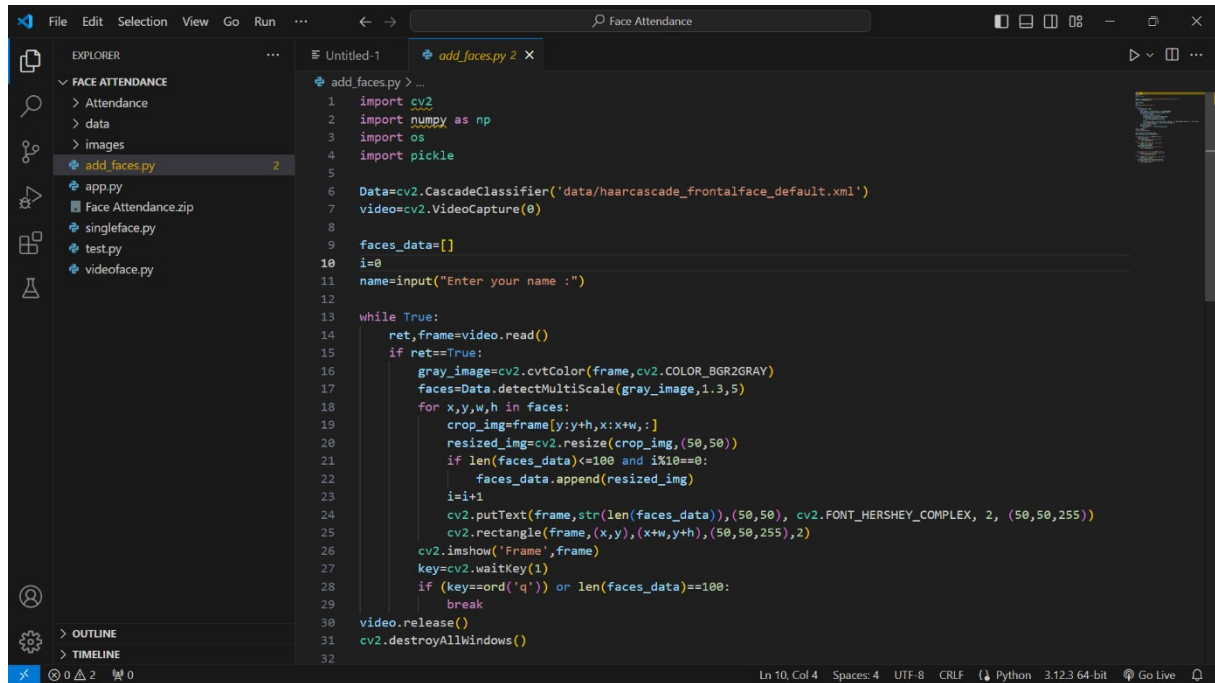
```
app.py > ...
1  import streamlit as st
2  import pandas as pd
3  from datetime import datetime
4  import time
5  ts=time.time()
6  date=datetime.fromtimestamp(ts).strftime("%d-%m-%y")
7  timestamp=datetime.fromtimestamp(ts).strftime("%H-%M-%S")
8
9  df=pd.read_csv("Attendance/Attendance_"+date+".csv")
10
11  st.dataframe(df.style.highlight_max(axis=0))
```

test.py is used to create models and recognize faces and store it in excel file.



```
test.py > ...
1  from sklearn.neighbors import KNeighborsClassifier
2  import cv2
3  import numpy as np
4  import os
5  import pickle
6  import csv
7  import time
8  from datetime import datetime
9
10  from win32com.client import Dispatch
11
12  def speak(str1):
13      speak=Dispatch(("SAPI.spVoice"))
14      speak.speak(str1)
15
16  Data=cv2.CascadeClassifier('data/haarcascade_frontalface_default.xml')
17  video=cv2.VideoCapture(0)
18
19
20
21  with open('data/names.pkl','rb') as w:
22      LABELS=pickle.load(w)
23  with open('data/face_data.pkl','rb') as f:
24      FACES=pickle.load(f)
25
26  print("Shape of face matrix :",FACES.shape)
27
28  COL_NAMES=['NAMES','TIME']
29
30  model=KNeighborsClassifier(n_neighbors=5)
31  model.fit(FACES,LABELS)
32
```

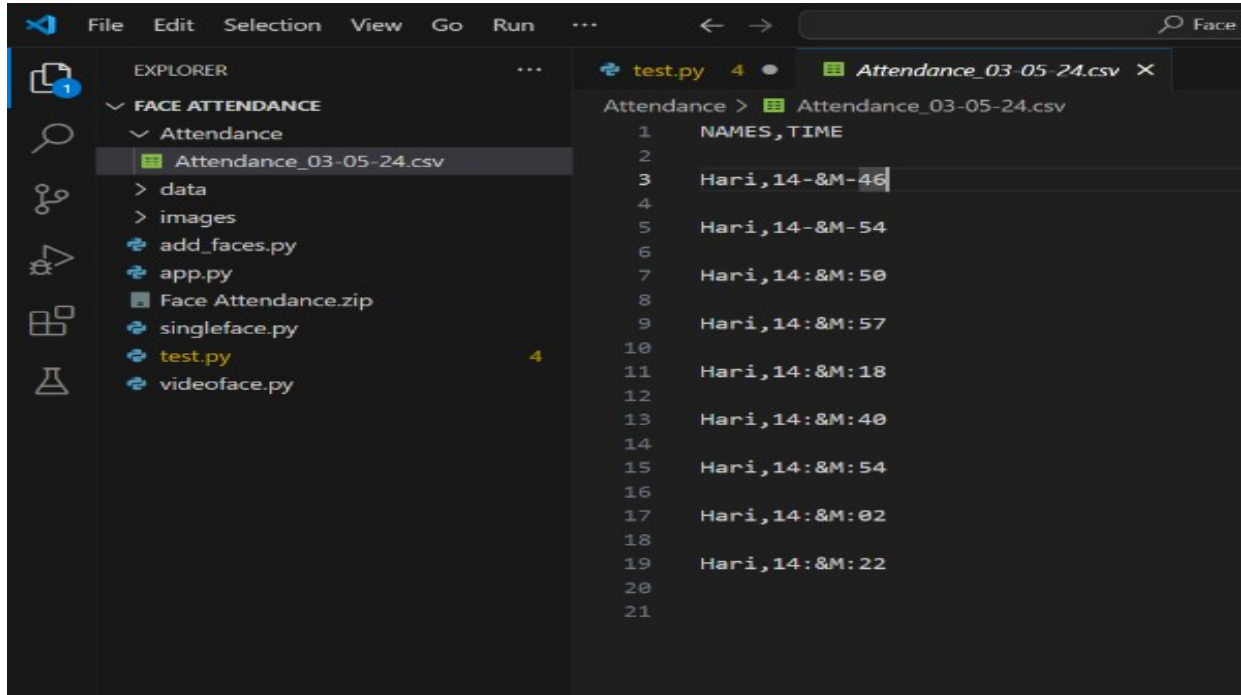
add_faces.py is used for collect faces through webcam and store it in pickle files



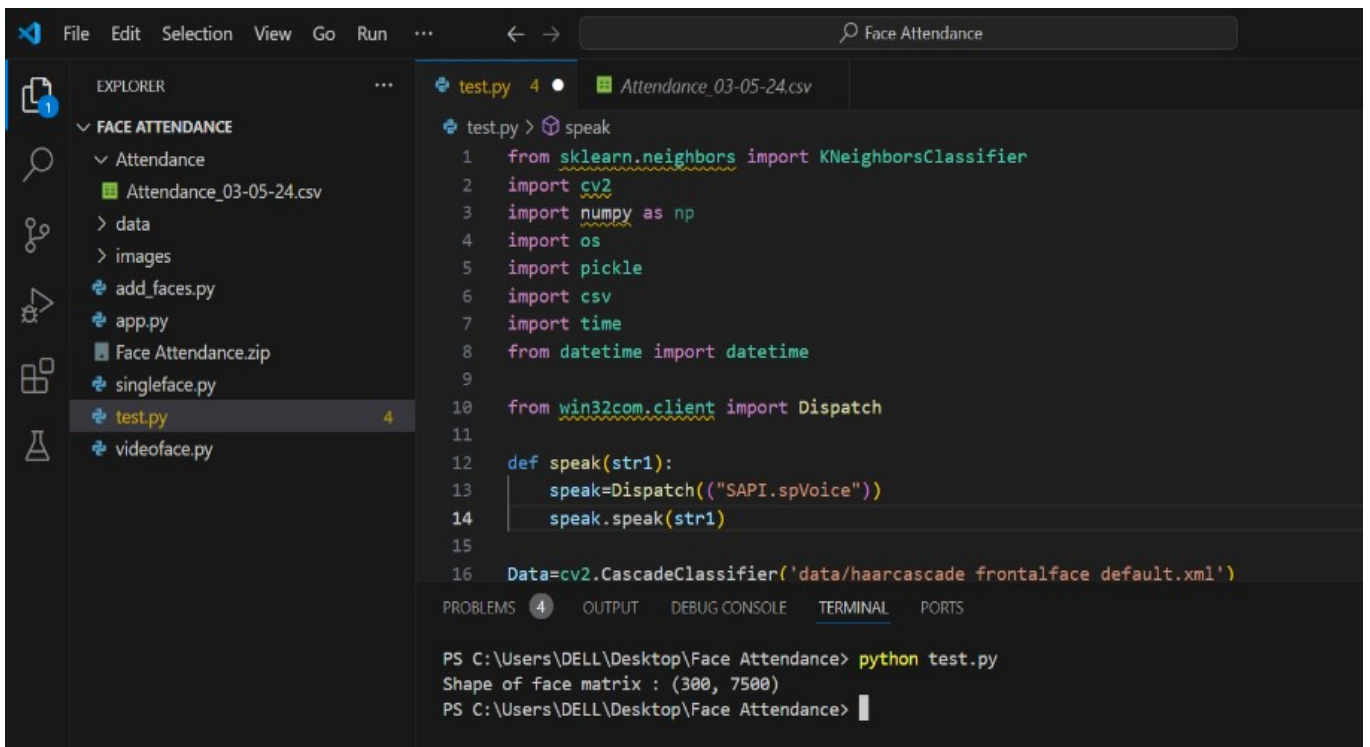
The image shows a screenshot of a Visual Studio Code editor window. The Explorer panel on the left shows a project named 'FACE ATTENDANCE' with subfolders 'Attendance', 'data', and 'images'. The 'add_faces.py' file is selected in the 'images' folder. The main editor area displays the code for 'add_faces.py'. The code imports cv2, numpy, os, and pickle. It initializes a CascadeClassifier with a default XML file, captures video from the webcam, and enters a loop to process frames. In each frame, it detects faces, crops them, resizes them to 50x50, and appends them to a list called 'faces_data'. It also displays the current count of faces and a bounding box around the detected face. The loop continues until the user presses 'q' or 100 faces are collected. Finally, it releases the video capture and destroys all windows.

```
1 import cv2
2 import numpy as np
3 import os
4 import pickle
5
6 Data=cv2.CascadeClassifier('data/haarcascade_frontalface_default.xml')
7 video=cv2.VideoCapture(0)
8
9 faces_data=[]
10 i=0
11 name=input("Enter your name :")
12
13 while True:
14     ret,frame=video.read()
15     if ret==True:
16         gray_image=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
17         faces=Data.detectMultiScale(gray_image,1.3,5)
18         for x,y,w,h in faces:
19             crop_img=frame[y:y+h,x:x+w,:]
20             resized_img=cv2.resize(crop_img,(50,50))
21             if len(faces_data)<=100 and i%10==0:
22                 faces_data.append(resized_img)
23             i=i+1
24             cv2.putText(frame,str(len(faces_data)),(50,50), cv2.FONT_HERSHEY_COMPLEX, 2, (50,50,255))
25             cv2.rectangle(frame,(x,y),(x+w,y+h),(50,50,255),2)
26             cv2.imshow('Frame',frame)
27             key=cv2.waitKey(1)
28             if (key==ord('q')) or len(faces_data)==100:
29                 break
30 video.release()
31 cv2.destroyAllWindows()
32
```

V. OUTPUTS:



```
Attendance_03-05-24.csv
1 NAMES, TIME
2
3 Hari, 14-&M-46
4
5 Hari, 14-&M-54
6
7 Hari, 14:&M:50
8
9 Hari, 14:&M:57
10
11 Hari, 14:&M:18
12
13 Hari, 14:&M:40
14
15 Hari, 14:&M:54
16
17 Hari, 14:&M:02
18
19 Hari, 14:&M:22
20
21
```



```
test.py
1 from sklearn.neighbors import KNeighborsClassifier
2 import cv2
3 import numpy as np
4 import os
5 import pickle
6 import csv
7 import time
8 from datetime import datetime
9
10 from win32com.client import Dispatch
11
12 def speak(str1):
13     speak=Dispatch(("SAPI.spVoice"))
14     speak.speak(str1)
15
16 Data=cv2.CascadeClassifier('data/haarcascade_frontalface_default.xml')
```

```
PS C:\Users\DELL\Desktop\Face Attendance> python test.py
Shape of face matrix : (300, 7500)
PS C:\Users\DELL\Desktop\Face Attendance>
```

The screenshot shows the Visual Studio Code interface. The Explorer panel on the left displays the project structure for 'FACE ATTENDANCE', including subfolders 'Attendance' and 'data', and files like 'Attendance_03-05-24.csv', 'add_faces.py', 'app.py', 'Face Attendance.zip', 'singleface.py', 'test.py', and 'videoface.py'. The main editor window shows the code for 'app.py', which imports 'streamlit' as 'st' and 'pandas' as 'pd'. It uses 'datetime' to get the current time and format it into a date and timestamp. The code then reads a CSV file from the 'Attendance' folder, sorts it by time, and displays the result using 'st.dataframe(df.style.highlight_max(axis=0))'. The bottom panel shows the 'TERMINAL' output, which includes the command to run 'test.py' (showing the shape of the face matrix as (300, 7500)) and the command to run 'app.py' using 'streamlit run'. It also provides the local URL 'http://localhost:8501' and the network URL 'http://192.168.1.5:8501'.

```
1 import streamlit as st
2 import pandas as pd
3 from datetime import datetime
4 import time
5 ts=time.time()
6 date=datetime.fromtimestamp(ts).strftime("%d-%m-%y")
7 timestamp=datetime.fromtimestamp(ts).strftime("%H-%M-%S")
8
9 df=pd.read_csv("Attendance/Attendance_"+date+".csv")
10
11 st.dataframe(df.style.highlight_max(axis=0))
```

PS C:\Users\DELL\Desktop\Face Attendance> python test.py
Shape of face matrix : (300, 7500)
PS C:\Users\DELL\Desktop\Face Attendance> python -m streamlit run app.py

You can now view your Streamlit app in your browser.

Local URL: <http://localhost:8501>
Network URL: <http://192.168.1.5:8501>

The screenshot shows a web browser window with the URL 'localhost:8501'. The browser displays the output of the Streamlit application, which is a table with three columns: 'NAMES' and 'TIME'. The table contains 10 rows of data. The last row, with index 9, is highlighted in yellow and shows 'sanji' and '16:&M:49'. The other rows show 'Hari' and various timestamps.

	NAMES	TIME
0	Hari	14:&M-46
1	Hari	14:&M-54
2	Hari	14:&M:50
3	Hari	14:&M:57
4	Hari	14:&M:18
5	Hari	14:&M:40
6	Hari	14:&M:54
7	Hari	14:&M:02
8	Hari	14:&M:22
9	sanji	16:&M:49

Network IP Address : [app · Streamlit](http://app.Streamlit)

VI. CONCLUSION :

The smart attendance system developed using machine learning in Python offers an efficient and accurate solution for automating attendance tracking through facial recognition technology. Leveraging Python libraries like OpenCV and Face Recognition, the system streamlines the process, eliminating manual entry and enhancing accuracy. Its versatility and open-source nature make it accessible and adaptable, holding promise for broader applications beyond attendance tracking, such as security and personalized experiences

VII. FUTURE SCOPE:

The future scope of automating attendance tracking through technologies like facial recognition extends beyond immediate efficiency gains to broader implications for security, data analytics, and user experience. As this technology evolves and becomes more widespread, several key advancements and applications can be anticipated.

Furthermore, the integration of facial recognition technology with existing systems like student information databases or HR management software will streamline administrative processes. This includes automatic updating of attendance records, generation of reports, and integration with payroll systems, thereby reducing manual data entry and potential errors.

In summary, the future scope of automating attendance tracking through facial recognition technology encompasses enhanced security, sophisticated data analytics, streamlined administrative processes, and continued advancements in biometric identification. These developments promise to revolutionize attendance management across diverse sectors, paving the way for more efficient, secure, and data-driven institutions and workplaces.