**punchbiz documentation**

SMART ATTENDANCE SYSTEM

TEAM MEMBERS:

CHANDRU T

SABARIGIRIVASAN S

**Face Attendance System**

1. **Introduction**

In today's fast-paced world, the need for efficient and accurate attendance tracking systems has become increasingly critical across various sectors, including education, corporate environments, and large-scale events. Traditional attendance methods, such as manual sign-ins or card-based systems, are often prone to errors, inefficiencies, and fraud. These challenges have paved the way for the development of advanced technologies aimed at automating and enhancing the attendance tracking process.

One promising solution is the implementation of a smart attendance system that leverages computer vision (CV), machine learning (ML), and face recognition algorithms. This system utilizes cutting-edge technology to recognize and verify individuals' faces in real-time, offering a seamless and non-intrusive method for attendance tracking. By harnessing the power of machine learning, the system continuously improves its accuracy and reliability, adapting to diverse conditions and environments.

The integration of these technologies into a user-friendly application is made possible through Streamlit, an open-source framework that allows for the rapid development and deployment of web-based data applications. Streamlit provides an intuitive interface for both administrators and users, simplifying the process of managing and interacting with the attendance system.

This smart attendance system not only enhances accuracy and efficiency but also offers numerous additional benefits. These include real-time monitoring, automated record keeping, and insightful analytics, all of which contribute to better resource management and improved operational workflows. As organizations continue to seek innovative solutions to streamline their operations, the adoption of advanced attendance systems powered by CV, ML, and face recognition stands out as a transformative approach that addresses the limitations of traditional methods.

**1.1 Project Overview**

The Face Attendance System is an innovative solution designed to streamline the process of recording attendance through the use of advanced facial recognition technology. Unlike traditional methods such as manual sign-ins or RFID cards, this system automates attendance tracking, thereby enhancing accuracy and efficiency while reducing administrative burden. The system captures and analyzes facial images to identify individuals, making it possible to mark attendance in real-time without manual intervention.

In educational institutions, corporate environments, and event management, keeping accurate attendance records is crucial. Traditional methods often suffer from issues like human error, time consumption, and potential for misuse. The Face Attendance System addresses these challenges by leveraging computer vision and machine learning to provide a seamless and reliable way of tracking attendance. By automatically identifying individuals based on their facial features, the system ensures that only authorized persons are marked present, thus preventing fraudulent entries and ensuring data integrity.

**1.2 Key Features**

**1.2.1. Automated Attendance:**

The Face Attendance System automates the process of attendance marking by using facial recognition. This eliminates the need for manual sign-ins or card swipes, saving time and reducing errors. Users simply need to look at a camera to have their attendance recorded, streamlining the entire process.

**1.2.2. Real-Time Recognition:**

The system provides real-time facial recognition, ensuring that attendance is marked as soon as an individual is detected by the camera. This capability is crucial for environments where quick and efficient processing is needed, such as during class transitions in schools or shift changes in workplaces.

**1.2.3. User-Friendly Interface:**

Designed with usability in mind, the system features an intuitive interface for both administrators and users. Administrators can easily manage user registrations, monitor attendance records, and configure system settings, while users benefit from a straightforward, hassle-free experience.

**1.2.4. Security:**

Ensuring the security of attendance data is a top priority. The Face Attendance System employs robust security measures, including data encryption and secure access controls, to protect against unauthorized access and data breaches. Compliance with data protection regulations such as GDPR is also considered, making the system suitable for use in environments with stringent privacy requirements.

**2. Requirements**

To successfully implement the Face Attendance System, the following software and hardware requirements must be met

**2.1 Software Requirements**

**Python 3.6+** : The system is built using Python, a versatile and powerful programming language.

**face-recognition==1.3.0** : A library for recognizing and manipulating faces.

**face-recognition-models==0.3.0** : Pre-trained models for face recognition.

**dlib** : A toolkit for making real-world machine learning and data analysis applications.

**distlib==0.3.8** : A library for distribution utilities.

**opencv-python==4.9.0.80** : A library for computer vision.

**cmake** : A build system required for compiling dlib.

**streamlit** : A framework for building web applications.

**dlib-19.24.1-cp311-cp311-win\_amd64.whl** : A precompiled binary for dlib for Windows.

**2.3 Installation Instructions**

**2.3.1. Clone the Repository:**

```bash

git clone https://github.com/yourusername/face-attendance-system.git

cd face-attendance-system

```

**2.3.2. Install Dependencies:**

```bash

pip install -r requirements.txt

```

**2.3.3. Install dlib:**

- For Windows, use the precompiled wheel:

```bash

pip install dlib-19.24.1-cp311-cp311-win\_amd64.whl

```

**2.3.4. Run the Application:**

```bash

streamlit run main.py

```

**3. Dataset Creation**

**3.1 Overview**

Creating a robust dataset of facial images is a crucial step in developing a reliable face recognition system. The dataset should consist of multiple photos of each individual, captured under different lighting conditions and angles to ensure the accuracy of the facial recognition model. This documentation provides a detailed guide on how to create a dataset using a webcam to capture photos.

Once collected, the data undergoes cleaning to remove duplicates, handle missing values, correct errors, and normalize formats. Afterward, data transformation steps such as feature engineering, scaling, and encoding are applied to prepare the dataset for analysis or machine learning tasks. This process ensures that the final dataset is high-quality, relevant, and useful for its intended purpose.

**3.2 Requirements**

Python 3.x

OpenCV library

Webcam

**3.3 Installation**

1. Install OpenCV:

```bash

pip install opencv-python

```

2. Create the Script: Save the following Python script as `capture\_photos.py`.

```python

import cv2

import os

def capture\_photos(folder\_name, num\_photos):

if not os.path.exists(folder\_name):

os.makedirs(folder\_name)

cap = cv2.VideoCapture(0)

face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')

photo\_count = 0

while photo\_count < num\_photos:

ret, frame = cap.read()

if not ret:

print("Failed to capture image")

continue

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)

for (x, y, w, h) in faces:

cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)

photo\_count += 1

photo\_name = f"{folder\_name}/photo\_{photo\_count}.jpg"

cv2.imwrite(photo\_name, frame)

print(f"Photo {photo\_count} captured and saved as {photo\_name}")

cv2.imshow('Captured Image', frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

folder\_name = 'Dataset\_New\\'

folder\_name += input("Enter the folder name to save photos: ")

num\_photos = 40

capture\_photos(folder\_name, num\_photos)

```

**3.4 Script Explanation**

3.4.1. Import Libraries:

```python

import cv2

import os

```

3.4.2 Define the `capture\_photos` Function:

- This function captures photos using a webcam and saves them in a specified folder.

- It first checks if the folder exists and creates it if not.

- It initializes the webcam and the Haar Cascade face detector.

- The function captures images in a loop until the specified number of photos is reached.

- Each captured image is converted to grayscale and detected using the Haar Cascade classifier.

- Detected faces are highlighted with rectangles, and the image is saved to the specified folder.

3.4.3 Capture and Save Photos:

The script captures images from the webcam and saves them as `photo\_X.jpg` in the specified folder, where `X` is the photo number.

It displays the captured image in a window and continues capturing until the specified number of photos is reached or the 'q' key is pressed.

**3.5 Running the Script**

1. Open a terminal or command prompt.

2. Navigate to the directory where `capture\_photos.py` is saved.

3. Run the script:

```bash

python capture\_photos.py

```

4. Enter the folder name where you want to save the photos when prompted.

**3.6 Usage Example**

When the script is executed, it will prompt you to enter a folder name. For example:

```

Enter the folder name to save photos: John\_Doe

```

The script will create a folder named `Dataset\_New\John\_Doe` and start capturing photos. It will capture 40 photos and save them in the specified folder. Each photo will be named `photo\_X.jpg` where `X` is the photo number.

**4.Encoding Images**

**4.1 Overview:**

After capturing photos, the next step is to encode these images. This involves processing each image to detect and encode the facial features. These encodings are then saved in a file, which will be used by the face recognition system to identify individuals. Below is the Python script that performs this task.

**Encode Images Script:**

```python

import face\_recognition

import os

import pickle

def save\_known\_faces(known\_face\_encodings, known\_face\_names, filename):

with open(filename, 'wb') as f:

pickle.dump((known\_face\_encodings, known\_face\_names), f)

def encode\_images\_from\_folder(folder\_path, encoding\_file):

known\_face\_encodings = []

known\_face\_names = []

for file\_name in os.listdir(folder\_path):

if file\_name.lower().endswith(('.jpg', '.jpeg', '.png', '.bmp')):

image\_path = os.path.join(folder\_path, file\_name)

print("Encoding:", file\_name)

try:

# Load the image

image = face\_recognition.load\_image\_file(image\_path)

# Find face locations using HOG model

face\_locations = face\_recognition.face\_locations(image, model="hog")

# If at least one face is found

if face\_locations:

# Encode the first face found in the image

face\_encoding = face\_recognition.face\_encodings(image, [face\_locations[0]])[0]

# Extract the name from the file name without extension

name = os.path.splitext(file\_name)[0]

known\_face\_encodings.append(face\_encoding)

known\_face\_names.append(name)

else:

print(f"No face found in {file\_name}")

except Exception as e:

print(f"Error processing {file\_name}: {e}")

save\_known\_faces(known\_face\_encodings, known\_face\_names, encoding\_file)

def main(folder\_path, encoding\_file):

encode\_images\_from\_folder(folder\_path, encoding\_file)

print("Encoding completed!")

# Specify folder path and encoding file here

folder\_path = "dataset"

encoding\_file = "known\_faces.pkl"

main(folder\_path, encoding\_file)

```

**5.Script Explanation**

**5.1 Import Libraries:**

```python

import face\_recognition

import os

import pickle

```

- `face\_recognition` is used for detecting and encoding faces.

- `os` is used for file and directory operations.

- `pickle` is used for saving and loading encoded face data.

**5.2 Define Functions:**

- `save\_known\_faces(known\_face\_encodings, known\_face\_names, filename)`:

- Saves the face encodings and names to a file using `pickle`.

- `encode\_images\_from\_folder(folder\_path, encoding\_file)`:

- Processes images in the specified folder to detect and encode faces.

- For each image, it detects faces and encodes the first face found.

- The face encodings and corresponding names (extracted from file names) are saved.

- `main(folder\_path, encoding\_file)`:

- Calls the encoding function with specified folder path and encoding file name.

- Prints a completion message.

**6.How to Use the Script:**

**6.1 Capture Photos:**

- Open a terminal or command prompt.

- Navigate to the directory where `capture\_photos.py` is saved.

- Run the script:

```bash

python capture\_photos.py

```

- Enter the folder name where you want to save the photos when prompted.

**6.2 Encode Images:**

- Open a terminal or command prompt.

- Navigate to the directory where `encode\_images\_from\_folder.py` is saved.

- Run the script:

```bash

python encode\_images\_from\_folder.py

```

- The script will process images in the specified folder, encode the faces,and save the encodings in file

**6.3 Conclusion**

This process ensures that you have a set of encoded facial features that the face recognition system can use to identify individuals. By capturing photos and encoding them, you create a foundation for reliable and accurate face recognition, which is essential for the Face Attendance System.

**7. Code : Recognizing Faces in an Image and Logging Attendance**

**7.1 Overview**

This code snippet loads known face encodings from a file, processes an image to recognize faces, annotates the recognized faces in the image, and logs the recognized names along with the timestamp to an Excel file.

**7.2 Detailed Steps**

7.2.1. Import Libraries:

- `face\_recognition`: For face detection and recognition.

- `numpy`: For numerical operations.

- `cv2` (OpenCV): For image processing.

- `os`: For file system operations.

- `pickle`: For loading and saving Python objects.

- `datetime`: For working with timestamps.

- `pandas`: For handling data and writing to Excel files.

- `matplotlib.pyplot`: For displaying images.

7.2.2. Load Image and Excel File Paths:

- `image\_path`: Path to the image in which faces need to be recognized.

- `excel\_filename`: Name of the Excel file where recognized names and timestamps will be saved.

7.2.3. Function to Load Known Faces:

```python

def load\_known\_faces(filename):

with open(filename, 'rb') as f:

known\_face\_encodings, known\_face\_names = pickle.load(f)

return known\_face\_encodings, known\_face\_names

```

- This function loads previously saved face encodings and corresponding names from a file using `pickle`.

7.2.4. Load Known Face Encodings and Names:

```python

known\_face\_encodings, known\_face\_names = load\_known\_faces('known\_faces.pkl')

```

7.2.5. Load the Image for Face Recognition:

```python

unknown\_image = face\_recognition.load\_image\_file(image\_path)

```

7.2.6. Detect Faces in the Image:

```python

face\_locations = face\_recognition.face\_locations(unknown\_image)

face\_encodings = face\_recognition.face\_encodings(unknown\_image, face\_locations)

```

7.2.7. Read the Image Using OpenCV:

```python

image = cv2.imread(image\_path)

```

7.2.8. Recognize Faces and Annotate the Image:

- Initialize an empty list to store recognized names and timestamps:

```python

recognized\_names = []

```

- Loop through each detected face location and encoding:

```python

for (top, right, bottom, left), face\_encoding in zip(face\_locations, face\_encodings):

```

- Compare the detected face encoding with known face encodings:

```python

matches = face\_recognition.compare\_faces(known\_face\_encodings, face\_encoding)

```

- Find the best match for the face:

```python

face\_distances = face\_recognition.face\_distance(known\_face\_encodings, face\_encoding)

best\_match\_index = np.argmin(face\_distances)

if matches[best\_match\_index]:

name = known\_face\_names[best\_match\_index]

recognized\_names.append((name, datetime.now()))

else:

name = "Unknown"

```

- Annotate the image with the recognized name:

```python

cv2.rectangle(image, (left, top), (right, bottom), (0, 0, 255), 2)

font = cv2.FONT\_HERSHEY\_DUPLEX

text\_size = cv2.getTextSize(name, font, 1, 2)[0]

cv2.rectangle(image, (left, bottom - text\_size[1] - 10), (right, bottom), (0, 0, 255), cv2.FILLED)

cv2.putText(image, name, (left + 10, bottom - 6), font, 1.0, (255, 255, 255), 1)

```

7.2.9. Save and Display the Annotated Image:

- Save the annotated image:

```python

cv2.imwrite('output\_image.png', image)

```

- Display the annotated image using Matplotlib:

```python

image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

plt.imshow(image\_rgb)

plt.axis('off')

plt.show()

```

7.2.10. Log Recognized Names and Timestamps to Excel:

- Create a DataFrame with recognized names and timestamps:

```python

df = pd.DataFrame(recognized\_names, columns=["Name", "Timestamp"])

```

- Append the data to an existing Excel file or create a new one if it doesn't exist:

```python

if os.path.exists(excel\_filename):

df.to\_excel(excel\_filename, index=False, header=False, mode='a')

else:

df.to\_excel(excel\_filename, index=False)

```

**7.3 Conclusion**

This code automates the process of face recognition in an image, annotates the image with recognized names, and logs the attendance details to an Excel file. It utilizes face recognition techniques, image processing, and data handling libraries to achieve this task efficiently.

**full code :**

import face\_recognition

import numpy as np

import cv2

import os

import pickle

from datetime import datetime

import pandas as pd

import matplotlib.pyplot as plt

# image\_path = r"Classroom images\IMG\_4336.jpg"

image\_path = r"WhatsApp Image 2024-04-24 at 19.43.04\_907cf64b.jpg"

excel\_filename = "recognized\_names.xlsx"

def load\_known\_faces(filename):

    with open(filename, 'rb') as f:

        known\_face\_encodings, known\_face\_names = pickle.load(f)

    return known\_face\_encodings, known\_face\_names

known\_face\_encodings, known\_face\_names = load\_known\_faces('known\_faces.pkl')

unknown\_image = face\_recognition.load\_image\_file(image\_path)

face\_locations = face\_recognition.face\_locations(unknown\_image)

face\_encodings = face\_recognition.face\_encodings(unknown\_image, face\_locations)

image = cv2.imread(image\_path)

recognized\_names = []

for (top, right, bottom, left), face\_encoding in zip(face\_locations, face\_encodings):

    matches = face\_recognition.compare\_faces(known\_face\_encodings, face\_encoding)

    name = "Unknown"

    face\_distances = face\_recognition.face\_distance(known\_face\_encodings, face\_encoding)

    best\_match\_index = np.argmin(face\_distances)

    if matches[best\_match\_index]:

        name = known\_face\_names[best\_match\_index]

        recognized\_names.append((name, datetime.now()))  # Appending name along with timestamp

    cv2.rectangle(image, (left, top), (right, bottom), (0, 0, 255), 2)

    font = cv2.FONT\_HERSHEY\_DUPLEX

    text\_size = cv2.getTextSize(name, font, 1, 2)[0]

    cv2.rectangle(image, (left, bottom - text\_size[1] - 10), (right, bottom), (0, 0, 255), cv2.FILLED)

    cv2.putText(image, name, (left + 10, bottom - 6), font, 1.0, (255, 255, 255), 1)

cv2.imwrite('output\_image.png', image)

image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

plt.imshow(image\_rgb)

plt.axis('off')  # Turn off axis

plt.show()

df = pd.DataFrame(recognized\_names, columns=["Name", "Timestamp"])

if os.path.exists(excel\_filename):

    df.to\_excel(excel\_filename, index=False, header=False, mode='a')

else:

    df.to\_excel(excel\_filename, index=False)

**8.Code : Real-Time Face Recognition Using Webcam or Video**

**8.1 Overview**

This script performs real-time face recognition using a webcam or a video file. It loads pre-encoded face data, captures video frames, detects faces, and identifies them by comparing with known faces.

**8.2 Detailed Steps**

8.2.1. Import Libraries:

- `os`, `cv2` (OpenCV): For file operations and image processing.

- `pickle`: For loading serialized face encodings.

- `numpy`: For numerical operations.

- `face\_recognition`: For face detection and recognition.

- `datetime`: For timestamps (if needed for attendance marking).

8.2.2. Load Known Faces:

```python

def load\_known\_faces(filename):

with open(filename, 'rb') as f:

known\_face\_encodings, known\_face\_names = pickle.load(f)

return known\_face\_encodings, known\_face\_names

```

- This function loads face encodings and corresponding names from a file using `pickle`.

8.2.3. Load Encodings and Class Names:

```python

encodeListKnown, classNames = load\_known\_faces('known\_faces.pkl')

print(len(encodeListKnown))

print('Encoding Complete')

```

- The function `load\_known\_faces` is called to get the encoded faces and names.

- Prints the number of known faces loaded and a completion message.

8.2.4. Initialize Video Capture:

```python

cap = cv2.VideoCapture(0) # Use 0 for webcam, or provide a file path for a video file

```

- Initializes video capture from the webcam (or from a video file if the path is provided).

8.2.5. Process Video Frames in a Loop:

```python

while True:

success, img = cap.read()

imgS = cv2.resize(img, (0, 0), None, 0.25, 0.25)

imgS = cv2.cvtColor(imgS, cv2.COLOR\_BGR2RGB)

```

- Captures a frame from the video.

- Resizes the frame to speed up face recognition (scales down by a factor of 4).

- Converts the frame from BGR to RGB color space, as required by `face\_recognition`.

8.2.6. Detect Faces and Encode Them:

```python

facesCurFrame = face\_recognition.face\_locations(imgS)

encodesCurFrame = face\_recognition.face\_encodings(imgS, facesCurFrame)

```

- Detects faces in the resized frame.

- Encodes the detected faces.

8.2.7. Compare Faces and Identify:

```python

for encodeFace, faceLoc in zip(encodesCurFrame, facesCurFrame):

matches = face\_recognition.compare\_faces(encodeListKnown, encodeFace)

faceDis = face\_recognition.face\_distance(encodeListKnown, encodeFace)

matchIndex = np.argmin(faceDis)

if matches[matchIndex]:

name = classNames[matchIndex].upper()

print(name)

```

- Iterates over each detected face and its encoding.

- Compares the encoded face with the known face encodings.

- Calculates the distance to find the best match.

- If a match is found, retrieves the corresponding name and prints it.

8.2.8. Annotate the Frame:

```python

y1, x2, y2, x1 = faceLoc

y1, x2, y2, x1 = y1 \* 4, x2 \* 4, y2 \* 4, x1 \* 4

cv2.rectangle(img, (x1, y1), (x2, y2), (140, 130, 50), 2)

cv2.rectangle(img, (x1, y2 - 35), (x2, y2), (140, 130, 50), cv2.FILLED)

cv2.putText(img, name, (x1 + 6, y2 - 6), cv2.FONT\_HERSHEY\_COMPLEX, 1, (255, 255, 255), 1)

```

- Draws a rectangle around the face.

- Draws a filled rectangle below the face rectangle for the name label.

- Puts the name text on the frame.

8.2.9. Display the Frame:

```python

cv2.imshow('Webcam', img)

if cv2.waitKey(1) == ord('q'):

break

```

- Displays the annotated frame.

- Breaks the loop and stops the script if the 'q' key is pressed.

8.2.10. Release Resources:

```python

cap.release()

cv2.destroyAllWindows()

```

- Releases the video capture object.

- Closes all OpenCV windows.

**8.3 Conclusion**

This script efficiently performs real-time face recognition by capturing video frames from a webcam, detecting and encoding faces, comparing them with known faces, annotating the frames, and displaying the results. It can be easily adapted for marking attendance or other applications where face recognition is required.

**full code :**

import os

import cv2

import pickle

import numpy as np

import face\_recognition

from datetime import datetime

last\_names = deque(maxlen=5)

def load\_known\_faces(filename):

    with open(filename, 'rb') as f:

        known\_face\_encodings, known\_face\_names = pickle.load(f)

    return known\_face\_encodings, known\_face\_names

encodeListKnown, classNames = load\_known\_faces('known\_faces.pkl')

print(len(encodeListKnown))

print('Encoding Complete')

# cap = cv2.VideoCapture("WhatsApp Video 2024-02-17 at 13.48.31\_a4439ec8.mp4")

cap = cv2.VideoCapture(0)

while True:

    success, img = cap.read()

    imgS = cv2.resize(img, (0, 0), None, 0.25, 0.25)

    imgS = cv2.cvtColor(imgS, cv2.COLOR\_BGR2RGB)

    facesCurFrame = face\_recognition.face\_locations(imgS)

    encodesCurFrame = face\_recognition.face\_encodings(imgS, facesCurFrame)

    for encodeFace, faceLoc in zip(encodesCurFrame, facesCurFrame):

        matches = face\_recognition.compare\_faces(encodeListKnown, encodeFace)

        faceDis = face\_recognition.face\_distance(encodeListKnown, encodeFace)

        matchIndex = np.argmin(faceDis)

        if matches[matchIndex]:

            name = classNames[matchIndex].upper()

            print(name)

            y1, x2, y2, x1 = faceLoc

            y1, x2, y2, x1 = y1 \* 4, x2 \* 4, y2 \* 4, x1 \* 4

            cv2.rectangle(img, (x1, y1), (x2, y2), (140, 130, 50), 2)

            cv2.rectangle(img, (x1, y2 - 35), (x2, y2), (140, 130, 50), cv2.FILLED)

            cv2.putText(img, name, (x1 + 6, y2 - 6), cv2.FONT\_HERSHEY\_COMPLEX, 1, (255,255), 1)

            # markAttendance(name)

    cv2.imshow('Webcame', img)

    if cv2.waitKey(1) == ord('q'):

        break

cap.release()

cv2.destroyAllWindows()

**9. Streamlit Application for Face Attendance System**

This code creates a Streamlit web application that allows users to upload an image, recognize faces in it, and download an attendance record.

**9.1 Code Explanation**

9.1.1. Import Libraries:

- `streamlit as st`: For creating the web application.

- `PIL.Image`: For handling image files.

- `face\_recognition`: For face detection and recognition.

- `numpy`, `cv2`, `os`, `pickle`: For numerical operations, image processing, file operations, and loading serialized data.

- `datetime`: For timestamps.

- `pandas as pd`: For data manipulation and exporting to CSV.

9.1.2. Function to Load Known Faces:

```python

def load\_known\_faces(filename):

with open(filename, 'rb') as f:

known\_face\_encodings, known\_face\_names = pickle.load(f)

return known\_face\_encodings, known\_face\_names

```

- Loads known face encodings and names from a specified file.

9.1.3. Function to Recognize Faces:

```python

def recognize\_faces(known\_face\_encodings, known\_face\_names, image\_path):

unknown\_image = face\_recognition.load\_image\_file(image\_path)

face\_locations = face\_recognition.face\_locations(unknown\_image)

face\_encodings = face\_recognition.face\_encodings(unknown\_image, face\_locations)

image = cv2.imread(image\_path)

recognized\_names = []

for (top, right, bottom, left), face\_encoding in zip(face\_locations, face\_encodings):

matches = face\_recognition.compare\_faces(known\_face\_encodings, face\_encoding)

name = "Unknown"

face\_distances = face\_recognition.face\_distance(known\_face\_encodings, face\_encoding)

best\_match\_index = np.argmin(face\_distances)

if matches[best\_match\_index]:

name = known\_face\_names[best\_match\_index]

recognized\_names.append((name, datetime.now()))

cv2.rectangle(image, (left, top), (right, bottom), (0, 0, 255), 2)

font = cv2.FONT\_HERSHEY\_DUPLEX

text\_size = cv2.getTextSize(name, font, 1, 2)[0]

cv2.rectangle(image, (left, bottom - text\_size[1] - 10), (right, bottom), (0, 0, 255), cv2.FILLED)

cv2.putText(image, name, (left + 10, bottom - 6), font, 1.0, (255, 255, 255), 1)

output\_image\_path = 'output\_image.png'

cv2.imwrite(output\_image\_path, image)

return recognized\_names, output\_image\_path

```

- Recognizes faces in an uploaded image, annotates the image, and returns the recognized names and the path to the annotated image.

9.1.4. Main Function for Streamlit Application:

```python

def main():

st.title("Face Attendance System")

image\_file = st.file\_uploader("Upload Image", type=["jpg", "jpeg", "png"])

if image\_file is not None:

image = Image.open(image\_file)

st.image(image, caption="Uploaded Image", use\_column\_width=True)

known\_face\_encodings, known\_face\_names = load\_known\_faces('known\_faces.pkl')

recognized\_names\_list, output\_image\_path = recognize\_faces(known\_face\_encodings, known\_face\_names, image\_file.name)

df = pd.DataFrame(recognized\_names\_list, columns=["Name", "Timestamp"])

st.image(output\_image\_path, caption="Predicted Image", use\_column\_width=True)

download\_button = st.download\_button(

label="Download Attendance",

data=df.to\_csv().encode(),

file\_name='Attendance\_taken.csv',

mime='text/csv'

)

if \_\_name\_\_ == "\_\_main\_\_":

main()

```

**9.2 Streamlit Application Layout:**

- Title: "Face Attendance System"

- File Uploader: Allows users to upload an image file.

- Displays the uploaded image.

- Loads known face encodings and names.

- Recognizes faces in the uploaded image and annotates it.

- Displays the annotated image.

- Provides a download button to download the attendance record as a CSV file.

**9.3 Conclusion**

This Streamlit application provides an easy-to-use interface for uploading images, recognizing faces, and downloading attendance records. It utilizes `face\_recognition` for face detection and recognition, and `pandas` for data handling.  
  
full code :

import streamlit as st

from PIL import Image

import face\_recognition

import numpy as np

import cv2

import os

import pickle

from datetime import datetime

import pandas as pd

def load\_known\_faces(filename):

    with open(filename, 'rb') as f:

        known\_face\_encodings, known\_face\_names = pickle.load(f)

    return known\_face\_encodings, known\_face\_names

def recognize\_faces(known\_face\_encodings, known\_face\_names, image\_path):

    unknown\_image = face\_recognition.load\_image\_file(image\_path)

    face\_locations = face\_recognition.face\_locations(unknown\_image)

    face\_encodings = face\_recognition.face\_encodings(unknown\_image, face\_locations)

    image = cv2.imread(image\_path)

    recognized\_names = []

    for (top, right, bottom, left), face\_encoding in zip(face\_locations, face\_encodings):

        matches = face\_recognition.compare\_faces(known\_face\_encodings, face\_encoding)

        name = "Unknown"

        face\_distances = face\_recognition.face\_distance(known\_face\_encodings, face\_encoding)

        best\_match\_index = np.argmin(face\_distances)

        if matches[best\_match\_index]:

            name = known\_face\_names[best\_match\_index]

            recognized\_names.append((name, datetime.now()))  # Appending name along with timestamp

        cv2.rectangle(image, (left, top), (right, bottom), (0, 0, 255), 2)

        font = cv2.FONT\_HERSHEY\_DUPLEX

        text\_size = cv2.getTextSize(name, font, 1, 2)[0]

        cv2.rectangle(image, (left, bottom - text\_size[1] - 10), (right, bottom), (0, 0, 255), cv2.FILLED)

        cv2.putText(image, name, (left + 10, bottom - 6), font, 1.0, (255, 255, 255), 1)

    output\_image\_path = 'output\_image.png'

    cv2.imwrite(output\_image\_path, image)

    return recognized\_names, output\_image\_path

def main():

    st.title("Face Attendance System")

    image\_file = st.file\_uploader("Upload Image", type=["jpg", "jpeg", "png"])

    if image\_file is not None:

        image = Image.open(image\_file)

        # st.image(image, caption="Uploaded Image", use\_column\_width=True)

        known\_face\_encodings, known\_face\_names = load\_known\_faces('known\_faces.pkl')

        recognized\_names\_list, output\_image\_path = recognize\_faces(known\_face\_encodings, known\_face\_names, image\_file.name)

        df = pd.DataFrame(recognized\_names\_list, columns=["Name", "Timestamp"])

        st.image(output\_image\_path, caption="Predicted Image", use\_column\_width=True)

        download\_button = st.download\_button(

            label="Download Attendance",

            data=df.to\_csv().encode(),

            file\_name='Attendance\_taken.csv',

            mime='text/csv'

        )

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**10. Conclusion:**

The implementation of a smart attendance system using computer vision, machine learning, face recognition algorithms, and Streamlit marks a significant advancement in automating and streamlining the attendance tracking process. By leveraging the power of face recognition, the system ensures a high level of accuracy and efficiency, mitigating common issues such as proxy attendance and manual errors associated with traditional methods. The integration of machine learning enhances the system’s capability to learn and adapt over time, improving recognition accuracy and robustness in varying conditions.

Computer vision plays a critical role in capturing and processing facial images in real-time, while machine learning algorithms ensure that the system can accurately distinguish between different faces, even in challenging scenarios. The use of Streamlit for the application interface provides an intuitive and user-friendly platform, making it accessible for administrators and users to manage and interact with the attendance system seamlessly.

This smart attendance system offers numerous benefits, including real-time attendance monitoring, automated record keeping, and the generation of insightful reports. These features contribute to increased productivity, better resource management, and enhanced security within an organization. Additionally, the scalability of the system allows it to be deployed across various sectors, such as education, corporate environments, and events, demonstrating its versatility and broad applicability.

In conclusion, the integration of computer vision, machine learning, and face recognition algorithms into a Streamlit application for attendance tracking represents a cutting-edge solution that addresses the limitations of traditional attendance systems. It enhances accuracy, efficiency, and user experience, positioning itself as a valuable tool for modern organizations aiming to leverage technology for operational excellence. As the system continues to evolve with advancements in AI and machine learning, it holds the potential to further revolutionize attendance management and other related domains.