Bit by bit explanation of the code

## Made by Aditya Singh (an AI/ML Aspirants)

### Functions and Their Explanations

#### `ensure\_directory\_and\_file\_exists`

```python

def ensure\_directory\_and\_file\_exists():

directory = attendance\_system\_path

if not os.path.exists(directory):

os.makedirs(directory)

csv\_path = os.path.join(directory, 'attendance.csv')

if not os.path.exists(csv\_path):

with open(csv\_path, 'w', newline='') as csvfile:

csvwriter = csv.writer(csvfile)

csvwriter.writerow(['Name', 'ID', 'Branch', 'Email', 'Date', 'Time'])

excel\_path = os.path.join(directory, 'attendance.xlsx')

if not os.path.exists(excel\_path):

df = pd.DataFrame(columns=['Name', 'ID', 'Branch', 'Email', 'Date', 'Time'])

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

train\_img\_path = os.path.join(directory, 'TrainingImage')

if not os.path.exists(train\_img\_path):

os.makedirs(train\_img\_path)

registered\_users\_path = os.path.join(directory, 'RegisteredUsers')

if not os.path.exists(registered\_users\_path):

os.makedirs(registered\_users\_path)

```

\*\*Explanation:\*\*

This function ensures that the necessary directories and files for storing training images, registered user images, and attendance records (both in CSV and Excel formats) are created if they don't already exist. It helps in maintaining the required folder structure for the application to function properly.

#### `encode\_image\_to\_base64`

```python

def encode\_image\_to\_base64(image\_path):

with open(image\_path, "rb") as image\_file:

encoded\_string = base64.b64encode(image\_file.read()).decode('utf-8')

return encoded\_string

```

\*\*Explanation:\*\*

This function reads an image from the given file path, encodes it to a base64 string, and returns the encoded string. This is useful for storing images in a text format within CSV or other files.

#### `decode\_base64\_to\_image`

```python

def decode\_base64\_to\_image(encoded\_string):

decoded\_image = base64.b64decode(encoded\_string)

image\_np = np.frombuffer(decoded\_image, dtype=np.uint8)

img = cv2.imdecode(image\_np, cv2.IMREAD\_COLOR)

return img

```

\*\*Explanation:\*\*

This function takes a base64 encoded string, decodes it back to binary data, converts it into a numpy array, and then decodes it into an image using OpenCV. This allows for the retrieval and display of images stored in a base64 format.

#### `clear\_entries`

```python

def clear\_entries():

txt\_id.delete(0, 'end')

txt\_name.delete(0, 'end')

txt\_branch.delete(0, 'end')

txt\_email.delete(0, 'end')

message\_label.config(text="")

clear\_camera\_frame()

```

\*\*Explanation:\*\*

This function clears the input fields (ID, Name, Branch, and Email) and any messages displayed on the GUI. It helps reset the input fields for new entries.

#### `clear\_camera\_frame`

```python

def clear\_camera\_frame():

camera\_label.config(image='')

```

\*\*Explanation:\*\*

This function clears the image displayed in the camera frame on the GUI. It's used to reset the camera frame when needed.

#### `update\_attendance\_for\_new\_day`

```python

def update\_attendance\_for\_new\_day():

csv\_path = os.path.join(attendance\_system\_path, 'attendance.csv')

try:

df = pd.read\_csv(csv\_path)

today\_date = datetime.now().strftime('%Y-%m-%d')

for index, row in df.iterrows():

last\_registered\_date = row['Date']

if last\_registered\_date != today\_date:

df.at[index, 'Date'] = today\_date

df.to\_csv(csv\_path, index=False)

except Exception as e:

print(f"Error updating attendance for new day: {str(e)}")

```

\*\*Explanation:\*\*

This function updates the attendance records to ensure that each user needs to mark their attendance again after 24 hours. It reads the current attendance records, checks the date, and updates the date if it is not the current date.

#### `save\_details\_to\_file`

```python

def save\_details\_to\_file(name, Id, branch, email, image\_path, time\_registered):

img = cv2.imread(image\_path)

cv2.putText(img, f"Name: {name}", (10, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 1)

cv2.putText(img, f"ID: {Id}", (10, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 1)

cv2.putText(img, f"Branch: {branch}", (10, 70), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 1)

cv2.putText(img, f"Email: {email}", (10, 90), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 1)

cv2.putText(img, f"Time Registered: {time\_registered}", (10, img.shape[0] - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 1)

encoded\_image = encode\_image\_to\_base64(image\_path)

csv\_path = os.path.join(attendance\_system\_path, 'attendance.csv')

with open(csv\_path, 'a', newline='') as file:

writer = csv.writer(file)

writer.writerow([name, Id, branch, email, datetime.now().strftime('%Y-%m-%d'), datetime.now().strftime('%H:%M:%S')])

excel\_path = os.path.join(attendance\_system\_path, 'attendance.xlsx')

df = pd.DataFrame({

'Name': [name],

'ID': [Id],

'Branch': [branch],

'Email': [email],

'Date': [datetime.now().strftime('%Y-%m-%d')],

'Time': [datetime.now().strftime('%H:%M:%S')]

})

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

with pd.ExcelWriter(excel\_path, engine='openpyxl', mode='a') as writer:

df.to\_excel(writer, index=False, header=False, startrow=len(pd.read\_excel(excel\_path)) + 1)

else:

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

save\_path = os.path.join(attendance\_system\_path, "RegisteredUsers")

if not os.path.exists(save\_path):

os.makedirs(save\_path)

cv2.imwrite(os.path.join(save\_path, f"{name}\_{Id}.jpg"), img)

```

\*\*Explanation:\*\*

This function saves user details and the captured image with overlaid text to CSV and Excel files. It also saves the image with the user's details on it to a specific folder. This ensures all user data, including images, is stored systematically.

#### `train\_images`

```python

def train\_images():

recognizer = cv2.face.LBPHFaceRecognizer\_create()

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

faces, Ids = get\_images\_and\_labels(attendance\_system\_path)

recognizer.train(faces, np.array(Ids))

recognizer.save(os.path.join(attendance\_system\_path, 'TrainingImageLabel/Trainner.yml'))

print("Training Complete")

```

\*\*Explanation:\*\*

This function trains the facial recognition model using the LBPH algorithm. It collects images and their corresponding labels, trains the model, and saves the trained model to a file. This allows for recognizing faces in future operations.

#### `get\_images\_and\_labels`

```python

def get\_images\_and\_labels(path):

image\_paths = [os.path.join(path, f) for f in os.listdir(path)]

faces = []

Ids = []

for image\_path in image\_paths:

if os.path.split(image\_path)[-1].split(".")[-1] == 'jpg':

pil\_image = Image.open(image\_path).convert('L')

image\_np = np.array(pil\_image, 'uint8')

Id = int(os.path.split(image\_path)[-1].split(".")[1])

faces.append(image\_np)

Ids.append(Id)

return faces, Ids

```

\*\*Explanation:\*\*

This function retrieves images and their labels from the specified directory. It reads the images, converts them to grayscale, extracts the ID from the filename, and stores the images and their IDs in separate lists. These lists are used to train the facial recognition model.

#### `take\_images`

```python

def take\_images():

Id = txt\_id.get()

name = txt\_name.get()

branch = txt\_branch.get()

email = txt\_email.get()

if Id == '' or name == '' or branch == '' or email == '':

message\_label.config(text="Please enter ID, Name, Branch, and Email")

else:

try:

Id = int(Id)

cam = cv2.VideoCapture(0)

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

sample\_num = 0

ensure\_directory\_and\_file\_exists()

```

\*\*Explanation:\*\*

This function captures images from the webcam, processes them to detect faces, and saves them along with the user's details. It ensures the necessary directories and files are

created before saving the images. This function is essential for registering new users and adding their images to the dataset.

### Summary

The provided functions collectively create a comprehensive face recognition attendance system. They handle tasks such as ensuring directory and file existence, encoding and decoding images, clearing input fields and camera frames, updating attendance for a new day, saving user details and images, training the facial recognition model, and capturing images for user registration. The system ensures that attendance is marked accurately and that user data is stored systematically, making it a robust solution for managing attendance using facial recognition.