CS4043D Image Processing

Course Project

Date: April 6 2024

Face Recognition: Attendance Tracking System

GROUP: D

Aaron Joseph	**B200047CS**
Abhay Raj	**B200014CS**
Anuram Anil	**B200731CS**
Harinarayanan J	**B200741CS**
Jackson Stephan	**B200743CS**

Department of Computer Science and Engineering

National Institute of Technology, Calicut

Declaration

This report has been prepared on the basis of our group's work. All the other published and unpublished source materials have been cited in this report.

Student Name:	**Signature:**
Aaron Joseph	[illegible]
Jackson Stephan	[illegible]
Harinarayanan J	[illegible]
Abhay Raj	[illegible]
Anuram Anil	[illegible]

Table of Contents

Abstract

Image processing involves the application of various algorithms and techniques to manipulate and analyse digital images. The process can enhance or extract useful information from images for further analysis or decision-making. Image processing has different applications in the real world ranging from image restoration, colour processing, compression to object recognition and pattern recognition. Under pattern recognition, facial recognition technology has taken the world by storm in recent years, revolutionizing various industries and applications. In the project, we explore the various methodologies when it comes to facial recognition and gain insights into the key components of a facial recognition system. The project takes inspiration from previous research papers, to use two feature extraction methods - Local Binary Pattern (LBP) and Histograms of Oriented Gradients (HOG) – and use a Random Forest Classifier to achieve the task of facial recognition and help create an attendance tracking system.

Project Specification

1. **Project Definition:** Utilizing image processing techniques to design a facial recognition model that can efficiently differentiate and correctly identify different faces in an image to help solve a real-life problem such as class attendance.

2. **Deliverables:**

   A software application that is capable of the following:

   a. Receive the image of the class as input

   b. Perform face detection and recognition from the image

   c. Compare the recognized faces with the pre-existing database and creating an attendance sheet.

3. **Features and Functionalities:**

   Face Recognition: The proposed system employs face detection and recognition using random forest and combination of LBP and HOG features. LBPH algorithm ensures to find the local structure of an image and it does that by comparing each pixel with its neighbouring pixels. We have to decide the number of neighbours to be considered for LBP and the sampling method. Each image is converted to grayscale before passing to LBP. LBP is applied on to each sub-region and a histogram of N bins is generated from the pixel labels. The individual histograms are concatenated into a single higher dimensional histogram.

   During recognition phase, using random forest (RF) classification, the results from each tree are collected for the input image and the majority voting is gathered to give the resulting class label. RF has been applied to compare the feature vectors from both training and testing stage to match the corresponding person.

   Attendance Marking: The application should automatically create an attendance sheet for recognized students based on the input image.

   Scalability: The system should be able to handle a growing number of students added to the facial database.

4. **Technical Specifications:**

   Hardware: Processing power sufficient for image processing and model execution.

   Software: Programming language with image processing libraries

Chapter 1: **Introduction**

1.1 Background

Facial Recognition involves the task of identification of a face from a video or image given a pre-existing database of faces. The process mainly consists of the detection of human faces from the image followed by the identification of the detected faces.

There has been substantial advancement in facial recognition technology due to the recent advancements in computer vision and machine learning algorithms. The applications of facial recognition spread out to multiple domains such as authentication, security and surveillance.

Image Processing methods play a crucial role in tasks like categorization, feature extraction and face detection. Some of the most common methods used in facial identification include Local Binary Patterns (LBP), Eigenfaces, Histogram of Oriented Gradients (HOG), and Convolutional Neural Networks (CNNs).

The Eigenfaces method goes through training images to try and extract components that catch maximum variance and discard the less important components. These extracted components are called principal components. Local Binary Patterns methods try to find a local structure of the image by comparing each pixel with its neighbouring pixels. Deep learning techniques like convolutional neural networks trained on massive datasets to achieve very high recognition rates.

Automated attendance systems are really useful in a variety of settings like offices, schools and public gatherings. The face recognition technology benefits in increased accuracy and efficiency when compared to conventional attendance methods.

1.2 Challenges

1. **Accuracy and Reliability:**

Achieving high accuracy and reliability is one of the primary challenges in facial recognition systems. Factors such as illumination variations, pose variations and occlusions could hinder the performance of the recognition system. Changes in lighting can significantly affect feature extraction, leading to false recognition. There is also a possibility that faces captured from various angles may not be recognized accurately. Occlusions like hats and scarves also make the process of recognition difficult. Hence developing a system from a robust algorithm is essential.

2. **Privacy Concerns:**

Significant privacy concerns arise, for facial recognition systems, regarding collection, storage and usage of data. Implementing appropriate security measures in compliance with privacy regulations is crucial.

3. **Dataset Management:**

Managing a large dataset of enrolled students facial image data poses challenges with respect to scalability, security and integrity of the system. The increase in student count would require efficient storage and mechanisms for retrieval.

Chapter 2: **Literature Review**

2.1 Recognition Using Eigen Faces and Artificial Neural Network [1]

The paper presents an approach to face recognition where the features are extracted from face images using principal component analysis (PCA) and derived using feed forward back propagation neural network (ANN). PCA searches a subspace of low dimension (face space) which has the maximum variance among a set of face images. The eigenvectors of the face space are called the eigenfaces, and a face image can be approximated by a linear combination of those eigenfaces. One ANN per person of the database was suggested, using the face descriptors (weights of eigenfaces) as inputs for training. The new face image is first projected into the face space and given as an input for each ANN. The maximum output determines the identity of the person. The proposed method was then subjected to testing using the ORL face database. The authors also compared their method with K-mean and Fuzzy Ant with Fuzzy C-means. The F.A.F.C obtained the recognition rate of 94.75%. The proposed method on the other hand obtained an improvement recognition rate of 97.018%.

Drawbacks:

1. Eigenface method is highly dependent on the number of eigenfaces used for feature extraction, and selecting an inappropriate number of eigenfaces can impact the recognition performance.

2. Computational complexity involved in determining the eigenvectors and eigenvalues, especially for large image sizes, can make the calculations challenging.

3. A face image can be approximately reconstructed by using its feature vector and the eigenfaces. But the degree of the fit or the "rebuild error ratio" increases as the training set members differ heavily from each other. This is due to the addition of the average face image that becomes messy.

4. Sensitivity to head orientations, as the model is prone to mismatches for images with large head orientations.

2.2 Face recognition and detection using Random Forest and combination of LBP and HOG [2]

This conference paper introduced a new framework to handle facial recognition and detection problem in a video broadcast under uncontrolled environments. It is based on an algorithm called Viola-Jones, which is a fast and accurate face detector locating faces in images and video sequences. Here a combination of LBP and HOG descriptors with a new formulation and new statistics extracting a feature vector robust and low-complexity are combined for face feature extraction resulting invariant to illumination, pose, expression and occlusion variations. Then an ensemble learning method called Random Forest used to face feature classification that presents high discrimination performance, low computational cost and outperformed other classifiers as SVM. In the voting stage a new formula is introduced to increase the accuracy of face verification, which compares the matching results from different frames of the video and selects the person with the highest percentage.

2.3 LBPH-based Enhanced Real-Time Face Recognition [3]

The paper proposes a facial recognition system based on the Local Binary Pattern Histogram (LBPH) method, which can handle low and high-resolution images, as well as variations in illumination, pose and occlusion. The paper describes the four main phases of the LBPH method: face detection, preprocessing, feature extraction and feature matching. The paper evaluates the performance of the LBPH method on a dataset of 1000 images of four subjects, and compares it with other methods such as SIFT, SURF, CNN, Gabor wavelet and PCA. It claims that the LBPH method achieves high accuracy and efficiency in face recognition.

Drawbacks:

1. The model may face challenges while trying to accurately recognize faces in scenarios with occlusion, pose variation, and illumination.

2. The document also mentions limitations in distance, algorithm maturity, and camera qualities that could affect the overall accuracy of the model.

Chapter 3: **Proposed Method**

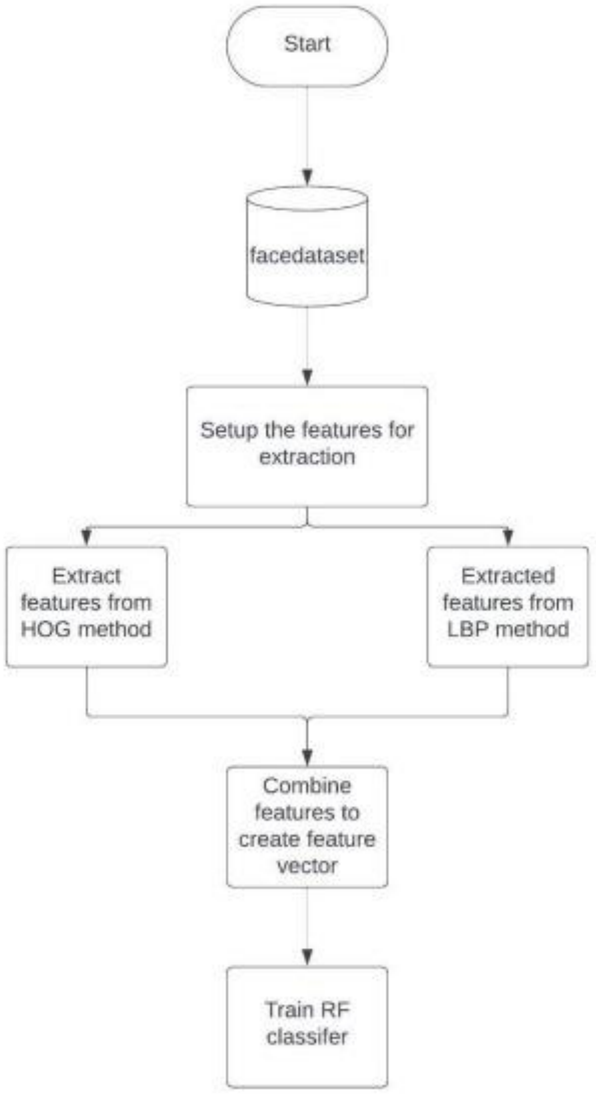

Start
facedataset
Setup the features for extraction
Extract features from HOG method
Extracted features from LBP method
Combine features to create feature vector
Train RF classifer

Figure 1. *Phases in Face Detection*

1. **Collection of Database**: For the purpose of creating the training database, 100 photos per subject are extracted for each training video, varying in terms of illumination, stance, background, expressions, and occlusions. After completing the face detection, the tagged faces are cropped off, leaving only the face, and the majority of the background is removed to create the training database. Following pre-processing, these faces are scaled, normalized, and turned to grayscale. The 3 sets of images that make up the testing data: the first set, which consists of ten images, is used to test the system's performance when there is only one person in the image; the second set is used to test the developed algorithm when there are multiple people in the image; and the third set is used when the person is not present in the training database.

2. **Face detection and Preprocessing image:** Face detection is a technique for locating faces in images and video sequences. A face detection algorithm called Viola & Jons is suggested. An image window with just the face region in it is the face detection process'

output. Following face detection and cropping of the face's windows, the histogram equalization approach is used to normalize the image's face.

3. **Features extraction**: The LBP and HOG methods combine two local feature approaches to extract features from a face classifier in order to identify or confirm the identity of an unknown face.

4. **Random Forest (RF) Classification module**: In order to match the appropriate person, RF has been used to compare extracted templates (vector features) from both the training and testing stages. To provide matching results, the features of the generated vectors from the testing data are compared to the vectors that were recorded in the training data.

3.1 Tools and Techniques

The implementation would be taking the help of the OpenCV library to use the built-in face recognizer - Local Binary Patterns (LBP) and Histograms Oriented Gradient (HOG) Face Recognizer.

LBP algorithm ensures to find the local structure of an image and it does that by comparing each pixel with its neighbouring pixels. We have to decide the number of neighbours to be considered for LBP and the sampling method. In our project we have decided a neighbourhood of 24 points within a set radius of 3 around the centre pixel. The method parameter is set to 'uniform', which means that it will consider only those patterns which have at most 2 bitwise transitions from 0 to 1 or vice versa when the bit pattern is traversed circularly. Each image is converted to grayscale before passing to LBP. LBP is applied on to each sub-region and a histogram of N “bins” is generated from the pixel labels. The “bins” parameter is set for each possible LBP value plus two extra bins for non-uniform patterns. The individual histograms are concatenated into a single higher dimensional histogram after normalization.

The HOG feature's computational complexity is significantly lower than that of the original data, and the HOG descriptor is resilient and insensitive to changes in geometry and illumination. The image will be divided into cells of 8x8 pixels size each. The cells are grouped into blocks of 2x2 cell size that are normalized together, which improves performance by providing some invariance to changes in lighting and shadowing.

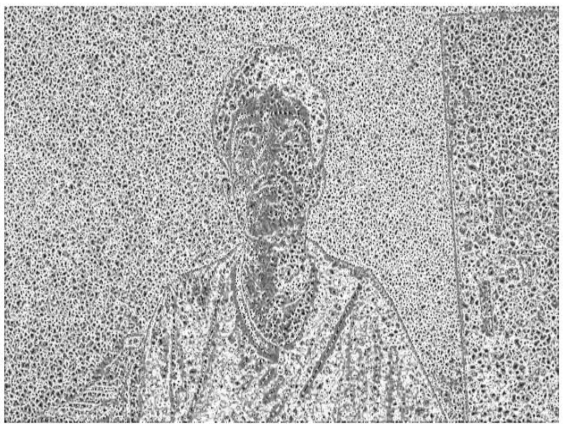
During recognition phase, using random forest (RF) classification, the features from LBP and HOG are combined to help create the decision trees. The results from each tree are collected for the input image and the majority voting is gathered to give the resulting class label. RF has been applied to compare the feature vectors from both training and testing stage to match the corresponding person.

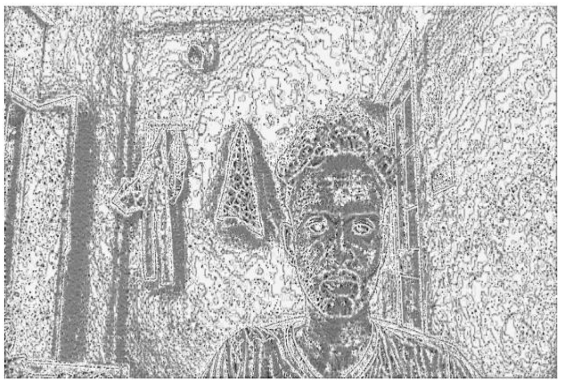
Chapter 4: **Evaluation and Results**

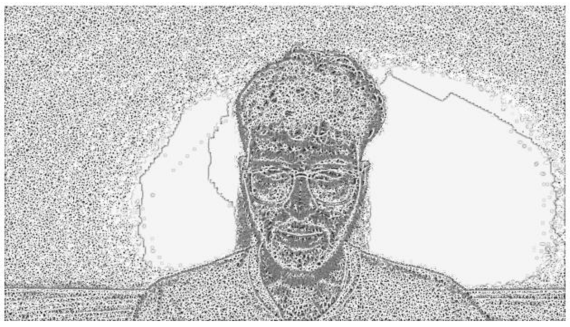
4.1 Evaluation

1. **LBP Faces**

The Local Binary Pattern (LBP) image of a digital image represents the texture information at each pixel. It's generated by comparing the intensity of a pixel with its neighboring pixels and encoding these comparisons into binary patterns.Here are a few LBP images for the images from the dataset:







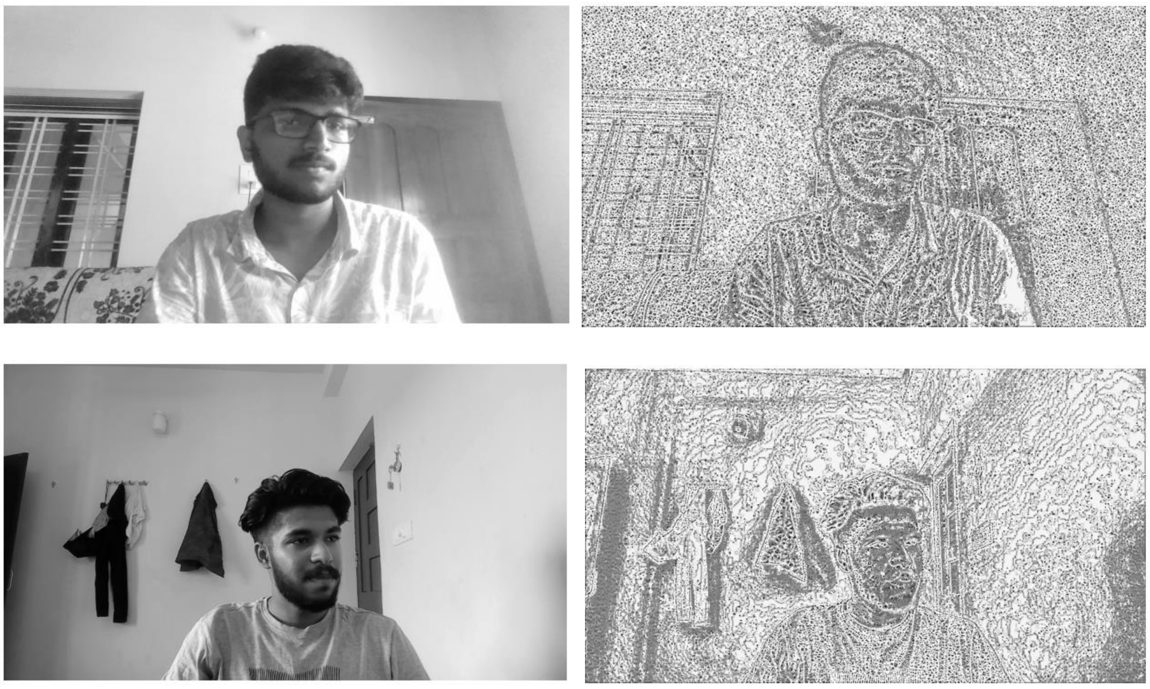
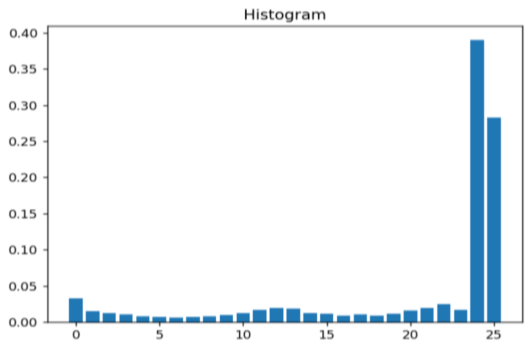


Figure 6: LBP images to the corresponding samples

2. **Histogram**

A histogram representation of an image provides a visual summary of the distribution of pixel intensities. It displays how many pixels in the image have specific intensity levels, ranging from 0 (black) to 255 (white) in grayscale images. Peaks in the histogram indicate areas of the image with high pixel concentration at certain intensity levels, while valleys represent regions with fewer pixels. Histograms are commonly used in image processing tasks like contrast adjustment, brightness correction, and thresholding to understand and manipulate the tonal distribution of an image. A few examples are shown below.



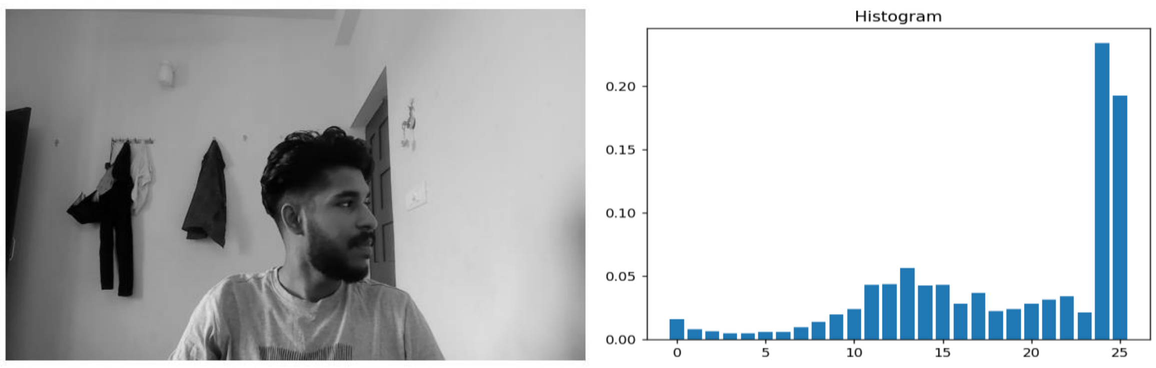


Figure 7:Histogram representations of the samples

4.2 Comparison of results

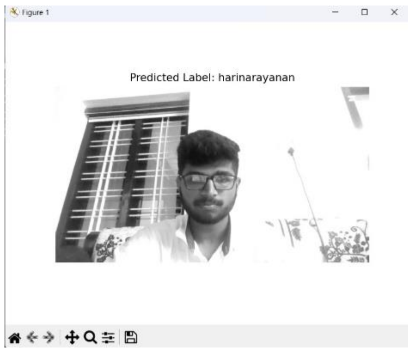
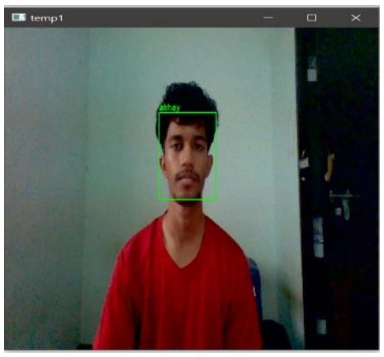
Local Binary Patterns (LBP) and Histogram of Oriented Gradients (HOG) are two widely used techniques in computer vision, particularly in tasks like face detection. LBPH (Local Binary Patterns Histograms) is a texture descriptor that encodes local texture information by comparing each pixel with its surrounding pixels and representing the result as a binary number. While LBPH is simple and computationally efficient, it may not capture high-level features well, limiting its accuracy in challenging conditions like variations in lighting and pose.

In contrast, the combination of LBP with HOG, referred to as LBP + HOG, offers a more comprehensive approach to feature representation. By integrating the local texture information from LBP with the global shape and edge information from HOG, this combined approach provides a more robust representation of complex characteristics like facial features. While LBP captures local texture details, HOG highlights global shape and edge features, resulting in a more nuanced and accurate representation of objects like faces.

However, the integration of LBP and HOG increases computational complexity compared to using either method alone. Despite this drawback, the combined approach tends to outperform LBPH alone, especially in challenging conditions where variations in lighting, pose, and facial expressions are common. The choice between LBPH and LBP + HOG depends on factors such as the desired level of accuracy, speed, and available computational resources.

Observation

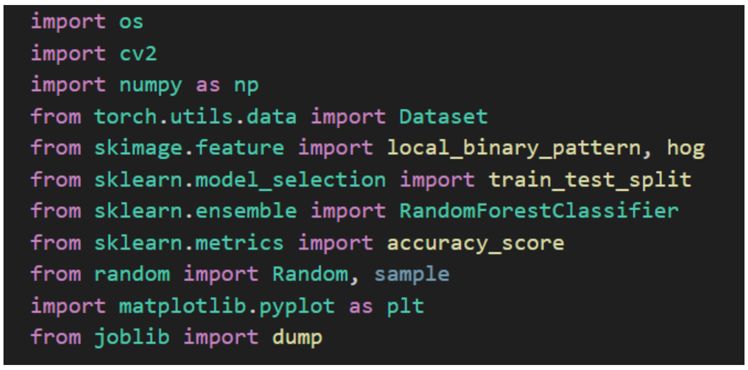
The LBPH model successfully identified one positive match but missed eight faces. It incorrectly identified one face. In contrast, the LBP + HOG model achieved seven correct identifications but made three false identifications.

Comparison of results	LBP and HOG Model	LBPH Model
Output	 Predicted Label: harinarayanan	
Time taken to train the model	9 mins	1.48 mins
Time taken to test	29.92 seconds (for 10 images in test_facedataset)	1.10 seconds (for 10 images in test_facedataset)
Test accuracy	70%	10%

Chapter 5: **Implementation and Code**

5.1 Documentation

<u>**File 1:**</u> **lbp_hog_train.py**

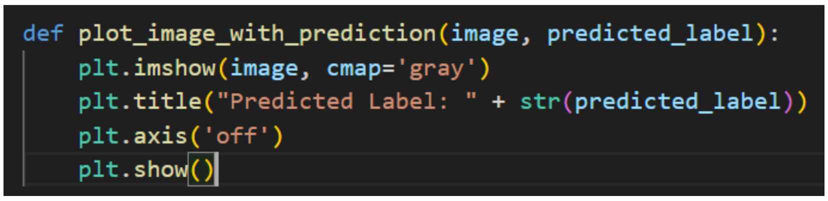


```
import os
import cv2
import numpy as np
from torch.utils.data import Dataset
from skimage.feature import local_binary_pattern, hog
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from random import Random, sample
import matplotlib.pyplot as plt
from joblib import dump
```

These are the necessary libraries imported for various tasks such as image processing (OpenCV), feature extraction (scikit-image), dataset handling (PyTorch), machine learning (scikit-learn), visualization (Matplotlib), and saving/loading models (joblib).

<u>**Function Definitions**</u>

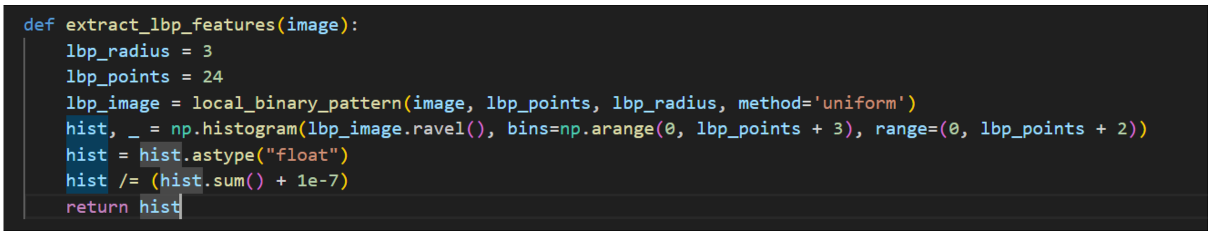
3. **plot_image_with_prediction(image, predicted_label):**



```
def plot_image_with_prediction(image, predicted_label):
    plt.imshow(image, cmap='gray')
    plt.title("Predicted Label: " + str(predicted_label))
    plt.axis('off')
    plt.show()
```

This function plots an image along with its predicted label.

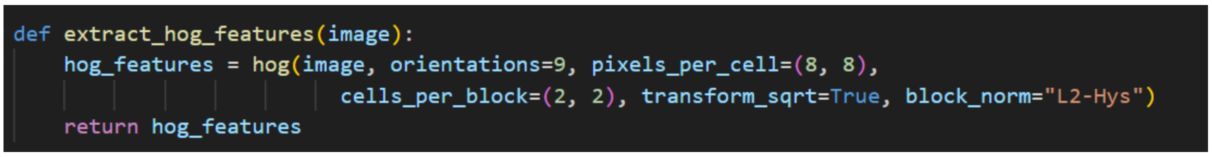
4. **extract_lbp_features(image):**



```
def extract_lbp_features(image):
    lbp_radius = 3
    lbp_points = 24
    lbp_image = local_binary_pattern(image, lbp_points, lbp_radius, method='uniform')
    hist, _ = np.histogram(lbp_image.ravel(), bins=np.arange(0, lbp_points + 3), range=(0, lbp_points + 2))
    hist = hist.astype("float")
    hist /= (hist.sum() + 1e-7)
    return hist
```

This function extracts Local Binary Pattern (LBP) features from the given grayscale image. *The explanation for each parameter in the above feature extraction is mentioned in tools and techniques.*

5. **extract_hog_features(image):**

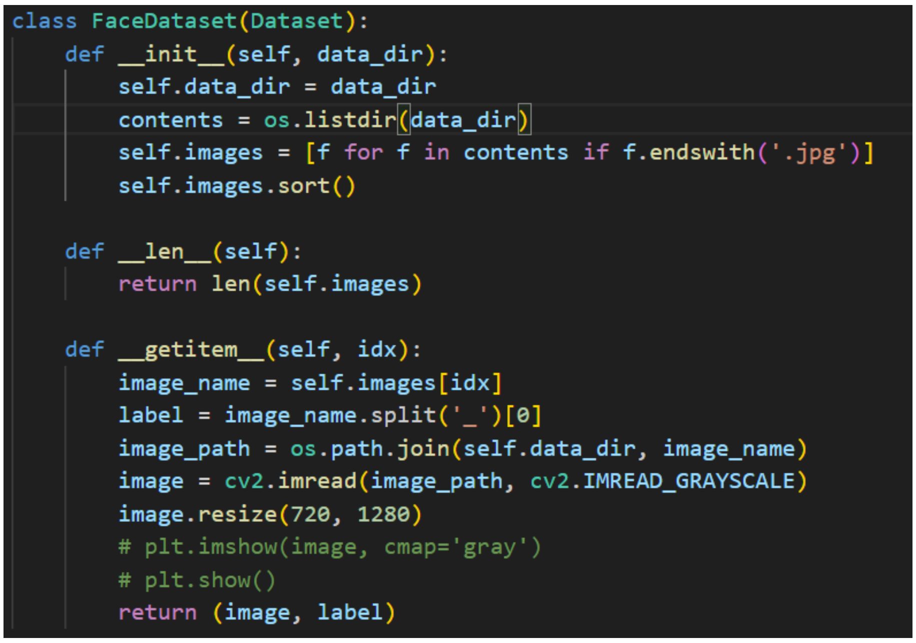


```
def extract_hog_features(image):
    hog_features = hog(image, orientations=9, pixels_per_cell=(8, 8),
                       cells_per_block=(2, 2), transform_sqrt=True, block_norm="L2-Hys")
    return hog_features
```

This function extracts Histogram of Oriented Gradients (HOG) features from the given grayscale image. *The explanation for each parameter in the above feature extraction is mentioned in tools and techniques.*

Class Definition

1. **class FaceDataset(Dataset):**



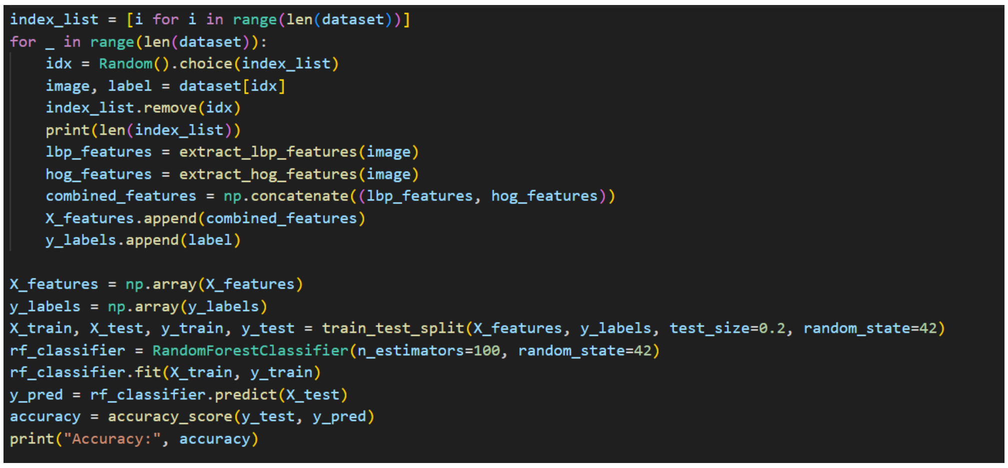
```
class FaceDataset(Dataset):
    def __init__(self, data_dir):
        self.data_dir = data_dir
        contents = os.listdir(data_dir)
        self.images = [f for f in contents if f.endswith('.jpg')]
        self.images.sort()

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        image_name = self.images[idx]
        label = image_name.split('_')[0]
        image_path = os.path.join(self.data_dir, image_name)
        image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        image.resize(720, 1280)
        # plt.imshow(image, cmap='gray')
        # plt.show()
        return (image, label)
```

The FaceDataset class is designed to facilitate the handling of a dataset containing face images. In its constructor method, it initializes by storing the directory path where the face images are located. It then lists all the files in the directory and filters out only the files with a '.jpg' extension. The filenames are sorted alphabetically to ensure consistent processing order.

The __len__ method returns the total number of images in the dataset. This method is called when using the len() function on an instance of FaceDataset, providing a convenient way to know the dataset's size.

The __getitem__ method retrieves an item from the dataset at a specified index. Given an index idx, it extracts the filename of the corresponding image and its label. Then, it constructs the full path to the image file and reads the image using OpenCV's cv2.imread() function in grayscale mode. The image is resized to a fixed size of 720x1280 pixels. Finally, it returns a tuple containing the grayscale image and its corresponding label.

Model Training and Evaluation



```
index_list = [i for i in range(len(dataset))]
for _ in range(len(dataset)):
    idx = Random().choice(index_list)
    image, label = dataset[idx]
    index_list.remove(idx)
    print(len(index_list))
    lbp_features = extract_lbp_features(image)
    hog_features = extract_hog_features(image)
    combined_features = np.concatenate((lbp_features, hog_features))
    X_features.append(combined_features)
    y_labels.append(label)

X_features = np.array(X_features)
y_labels = np.array(y_labels)
X_train, X_test, y_train, y_test = train_test_split(X_features, y_labels, test_size=0.2, random_state=42)
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)
y_pred = rf_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

The provided code block is responsible for training a Random Forest classifier, for face recognition/ classification tasks.

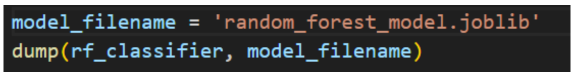
Initially, it shuffles the dataset and extracts features from each face image using Local Binary Pattern (LBP) and Histogram of Oriented Gradients (HOG) techniques. This process ensures diverse feature representation for each image, enhancing the model's ability to learn discriminative patterns. The features are concatenated into a single feature vector for each image.

Subsequently, the dataset is split into training and testing sets using the *train_test_split()* function from scikit-learn. This step is crucial for assessing the model's performance on unseen data and avoiding overfitting.

A Random Forest classifier is then instantiated with 100 decision trees and trained on the extracted features and corresponding labels using the *fit()* method. Random Forest is a powerful ensemble learning method capable of handling high-dimensional data and capturing complex relationships between features and labels.

After training, the model's accuracy is evaluated on the testing set using the *accuracy_score()* function from scikit-learn. The accuracy represents the proportion of correctly classified samples in the testing set, providing insight into the model's performance.

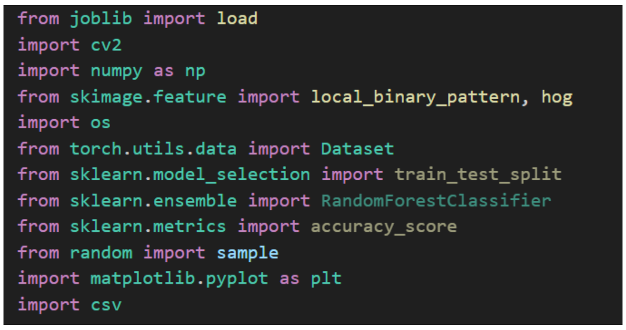
Saving the Trained Model



```
model_filename = 'random_forest_model.joblib'
dump(rf_classifier, model_filename)
```

The trained classifier is saved to a file using *dump()* function from joblib library.

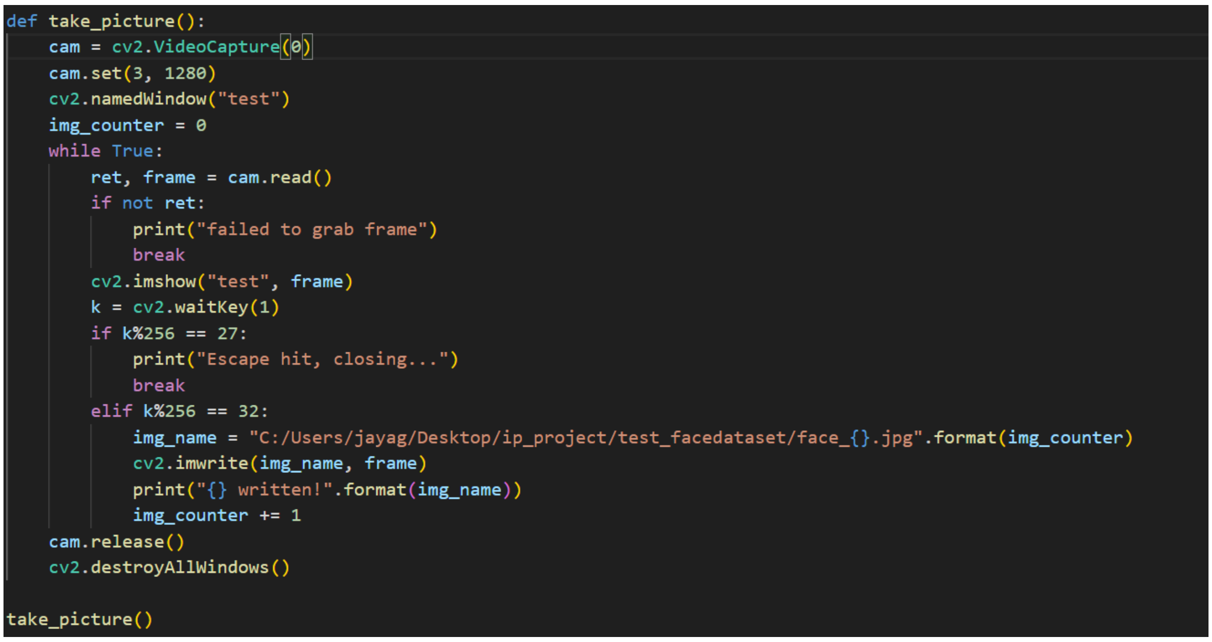
<u>File 2:</u> lbp_hog_test.py



```python
from joblib import load
import cv2
import numpy as np
from skimage.feature import local_binary_pattern, hog
import os
from torch.utils.data import Dataset
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from random import sample
import matplotlib.pyplot as plt
import csv
```

This code imports necessary libraries and modules for conducting face recognition, including OpenCV for image processing, scikit-image for feature extraction, and scikit-learn for machine learning functionalities. Additionally, it imports modules for handling datasets and data splitting, visualization, and CSV file operations.

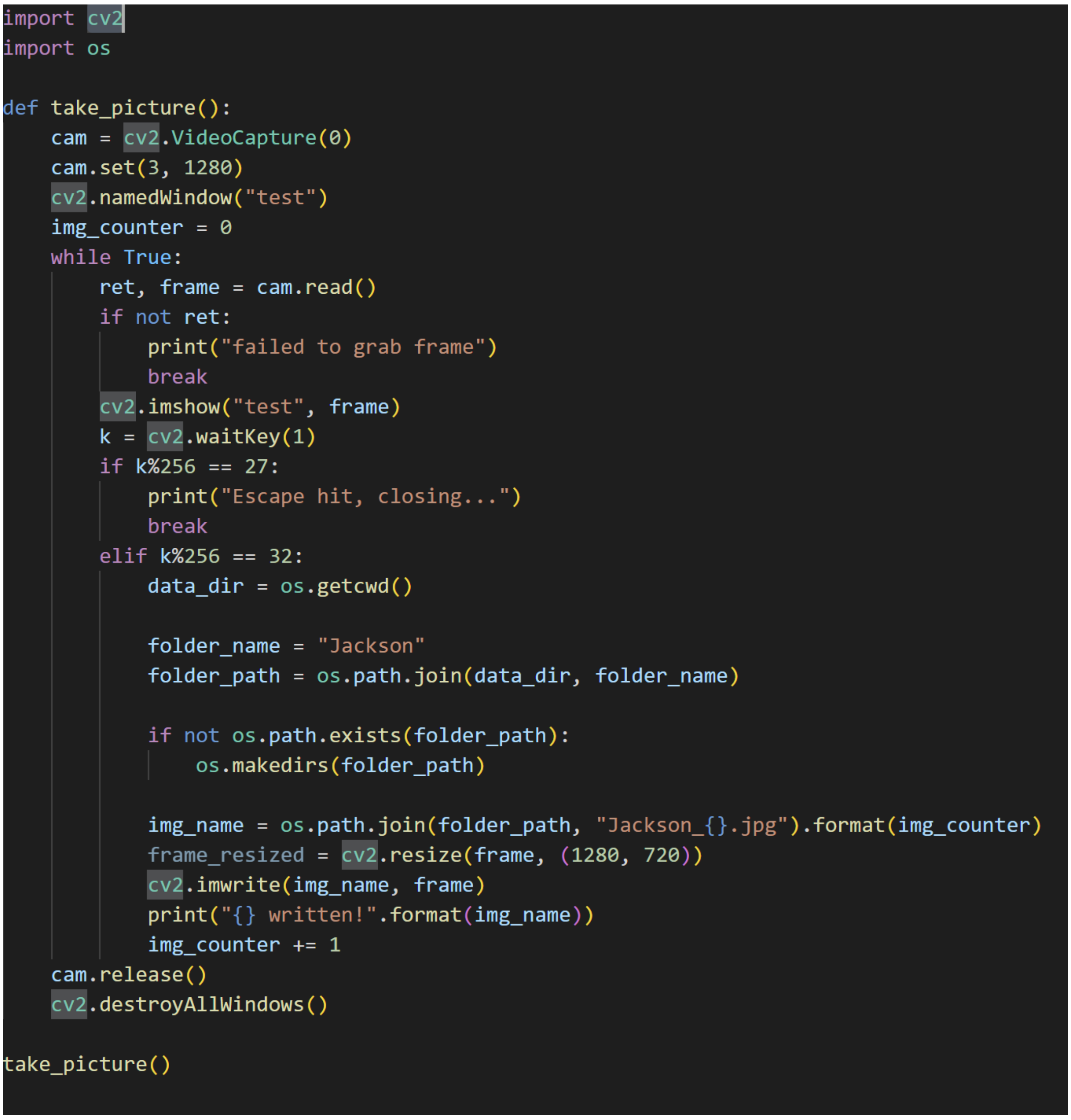
Image capturing for test_facedataset



```python
def take_picture():
    cam = cv2.VideoCapture(0)
    cam.set(3, 1280)
    cv2.namedWindow("test")
    img_counter = 0
    while True:
        ret, frame = cam.read()
        if not ret:
            print("failed to grab frame")
            break
        cv2.imshow("test", frame)
        k = cv2.waitKey(1)
        if k%256 == 27:
            print("Escape hit, closing...")
            break
        elif k%256 == 32:
            img_name = "C:/Users/jayag/Desktop/ip_project/test_facedataset/face_{}.jpg".format(img_counter)
            cv2.imwrite(img_name, frame)
            print("{} written!".format(img_name))
            img_counter += 1
    cam.release()
    cv2.destroyAllWindows()

take_picture()
```

The *take_picture()* function allows users to capture images from a webcam and save them for further processing. Upon execution, it initializes the webcam capture using OpenCV's *cv2.VideoCapture(0)* function, setting the resolution to 1280x720 pixels. It creates a window named "test" to display the live video feed.

We use student images as a dataset. We label each student's image with their names to enable the model to extract the label. We train the model using 100 images of each student, which we store in a folder named "facedataset." The transcription will require almost 5–10 minutes for completion. The images are required to have a dimension of 1280x720. We acquired the data using a Python script that enables users to capture images in the required format. The code also assigns a label to the captured images.



```python
import cv2
import os

def take_picture():
    cam = cv2.VideoCapture(0)
    cam.set(3, 1280)
    cv2.namedWindow("test")
    img_counter = 0
    while True:
        ret, frame = cam.read()
        if not ret:
            print("failed to grab frame")
            break
        cv2.imshow("test", frame)
        k = cv2.waitKey(1)
        if k%256 == 27:
            print("Escape hit, closing...")
            break
        elif k%256 == 32:
            data_dir = os.getcwd()

            folder_name = "Jackson"
            folder_path = os.path.join(data_dir, folder_name)

            if not os.path.exists(folder_path):
                os.makedirs(folder_path)

            img_name = os.path.join(folder_path, "Jackson_{}.jpg").format(img_counter)
            frame_resized = cv2.resize(frame, (1280, 720))
            cv2.imwrite(img_name, frame)
            print("{} written!".format(img_name))
            img_counter += 1
    cam.release()
    cv2.destroyAllWindows()

take_picture()
```

1. Importing required libraries:
   - cv2: This is the OpenCV library used for computer vision tasks, including image processing and video capture.
   - os: This module provides functions to interact with the operating system, allowing operations such as file manipulation.
2. Define the take_picture() function:
   - This function captures images from the webcam and saves them to a specified folder.
3. Inside take_picture() function:
   - Initialize the webcam using cv2.VideoCapture(0). Here, 0 denotes the default webcam device. You can specify a different index if you have multiple cameras connected.
   - Set the resolution of the camera using cam.set(3, 1280), where 3 corresponds to the property ID for width.
   - Create a named window called "test" using cv2.namedWindow("test") for displaying the captured frames.
   - Initialize a counter img_counter to keep track of the number of images captured.
4. Start an infinite loop to continuously capture frames from the webcam:
   - Use cam.read() to capture a frame. ret will be True if the frame is successfully captured.
   - Display the captured frame using cv2.imshow("test", frame).

- Wait for a key press using cv2.waitKey(1).
- If the Escape key (ASCII value 27) is pressed, break out of the loop and release the webcam resources using cam.release(). Also, close all OpenCV windows using cv2.destroyAllWindows().
- If the Space key (ASCII value 32) is pressed, proceed to save the captured frame:
  - Determine the current working directory using os.getcwd().
  - Define the folder name where the images will be saved (in this case, "Jackson").
  - Create the folder if it doesn't exist using os.makedirs(folder_path).
  - Generate the image name based on the counter (img_counter) and save the frame to the specified folder using cv2.imwrite().
  - Increment the counter img_counter.
  - Print the path of the saved image.
  - The image is assigned the label (in this case Jackson_1..n)

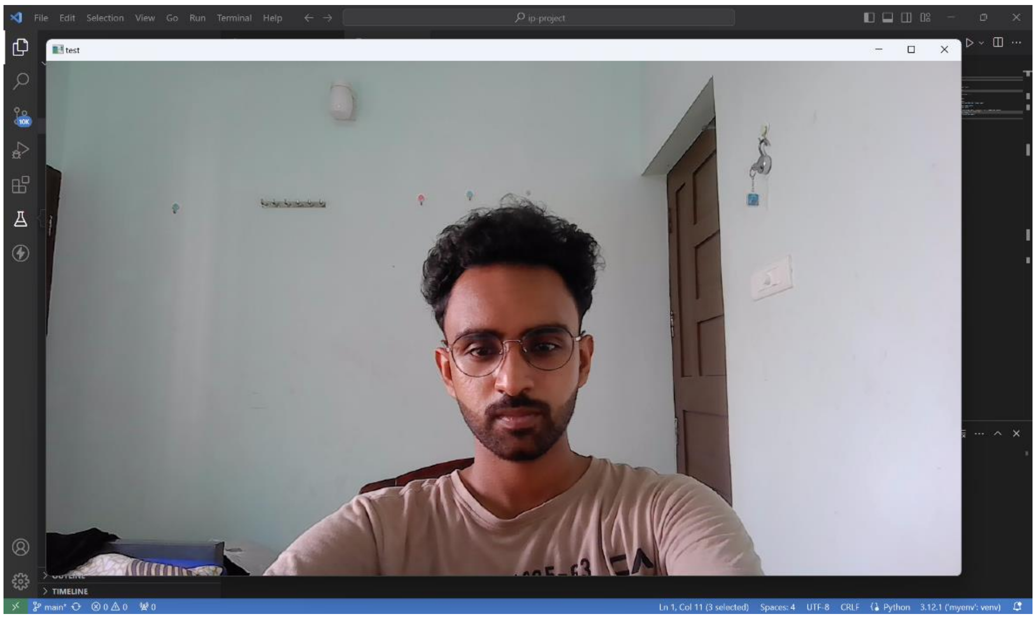
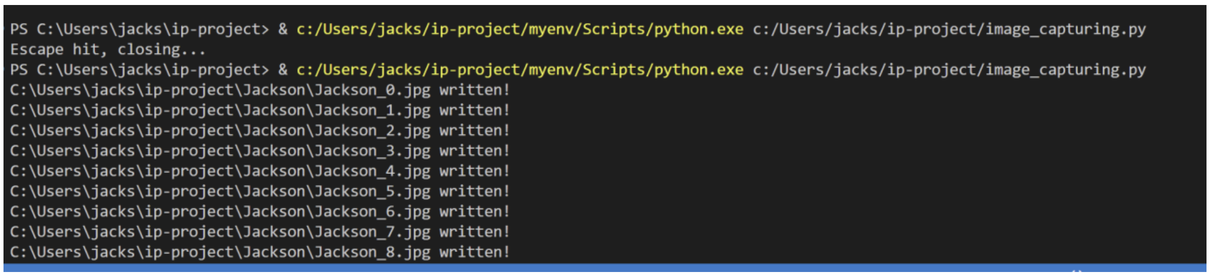


Figure 1: Camera activated when the code is activated

5. After capturing images or exiting the loop, release the webcam resources and close the OpenCV windows.



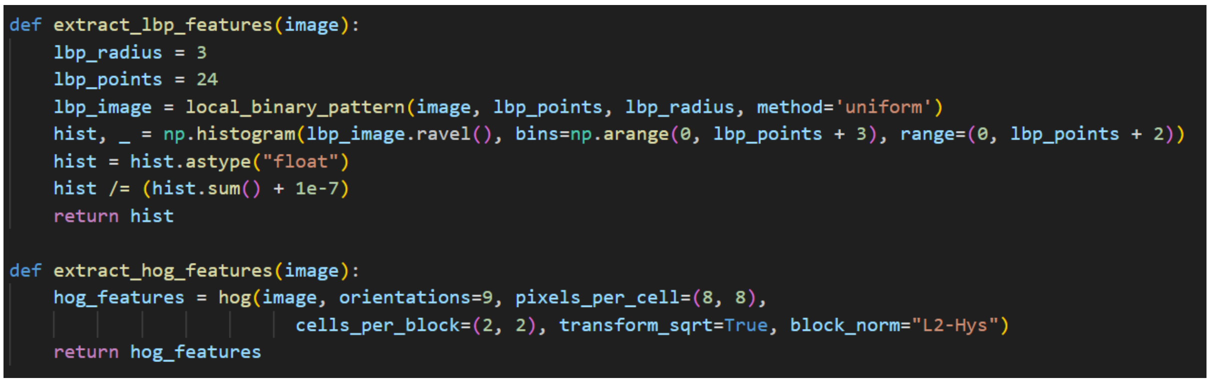
```
PS C:\Users\jacks\ip-project> & c:/Users/jacks/ip-project/myenv/Scripts/python.exe c:/Users/jacks/ip-project/image_capturing.py
Escape hit, closing...
PS C:\Users\jacks\ip-project> & c:/Users/jacks/ip-project/myenv/Scripts/python.exe c:/Users/jacks/ip-project/image_capturing.py
C:\Users\jacks\ip-project\Jackson\Jackson_0.jpg written!
C:\Users\jacks\ip-project\Jackson\Jackson_1.jpg written!
C:\Users\jacks\ip-project\Jackson\Jackson_2.jpg written!
C:\Users\jacks\ip-project\Jackson\Jackson_3.jpg written!
C:\Users\jacks\ip-project\Jackson\Jackson_4.jpg written!
C:\Users\jacks\ip-project\Jackson\Jackson_5.jpg written!
C:\Users\jacks\ip-project\Jackson\Jackson_6.jpg written!
C:\Users\jacks\ip-project\Jackson\Jackson_7.jpg written!
C:\Users\jacks\ip-project\Jackson\Jackson_8.jpg written!
```

Figure 2: The images being saved to the folder with a proper label

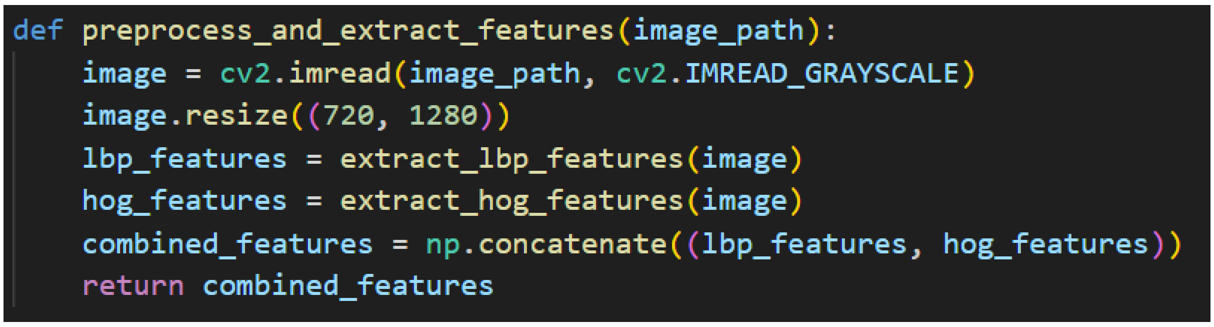
Inside the main loop, it continuously reads frames from the webcam using *cam.read()* and displays them in the "test" window using *cv2.imshow()*. It waits for key events using *cv2.waitKey(1)*. If the 'Esc' key (ASCII value 27) is pressed, indicating the user wants to exit, the loop breaks, and the webcam capture is released using *cam.release()*, and the OpenCV windows are destroyed with *cv2.destroyAllWindows()*.

If the 'Space' key (ASCII value 32) is pressed, it saves the current frame as an image file with a unique name based on the value of `img_counter`. The image is saved in the specified directory as a JPEG file with the naming convention "face_{}.jpg", where `{}` is replaced by the current value of `img_counter`. Finally, `img_counter` is incremented to ensure each image has a unique filename.

Feature Extraction



```python
def extract_lbp_features(image):
    lbp_radius = 3
    lbp_points = 24
    lbp_image = local_binary_pattern(image, lbp_points, lbp_radius, method='uniform')
    hist, _ = np.histogram(lbp_image.ravel(), bins=np.arange(0, lbp_points + 3), range=(0, lbp_points + 2))
    hist = hist.astype("float")
    hist /= (hist.sum() + 1e-7)
    return hist

def extract_hog_features(image):
    hog_features = hog(image, orientations=9, pixels_per_cell=(8, 8),
                          cells_per_block=(2, 2), transform_sqrt=True, block_norm="L2-Hys")
    return hog_features
```



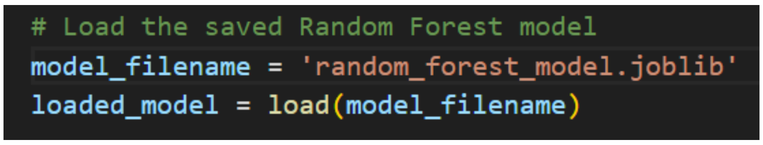
```python
def preprocess_and_extract_features(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    image.resize((720, 1280))
    lbp_features = extract_lbp_features(image)
    hog_features = extract_hog_features(image)
    combined_features = np.concatenate((lbp_features, hog_features))
    return combined_features
```

The function extract_lbp_features(image) computes Local Binary Pattern (LBP) features from a given grayscale image. It first defines parameters such as the radius (lbp_radius) and the number of points (lbp_points) for the LBP calculation. Using the *local_binary_pattern()* function from the skimage.feature module, it generates the LBP image representation of the input image. Then, it constructs a histogram (hist) of the LBP image intensities, ensuring that the bins cover the range of possible intensity values. The histogram is normalized by dividing each bin count by the total count to obtain normalized LBP features. Finally, it returns the normalized histogram representing the LBP features of the input image.

The function extract_hog_features(image) computes Histogram of Oriented Gradients (HOG) features from a given grayscale image. It utilizes the *hog()* function from the skimage.feature module to calculate the HOG features.

The function preprocess_and_extract_features(image_path) performs preprocessing and feature extraction on a given image file located at image_path and returns the combined features.

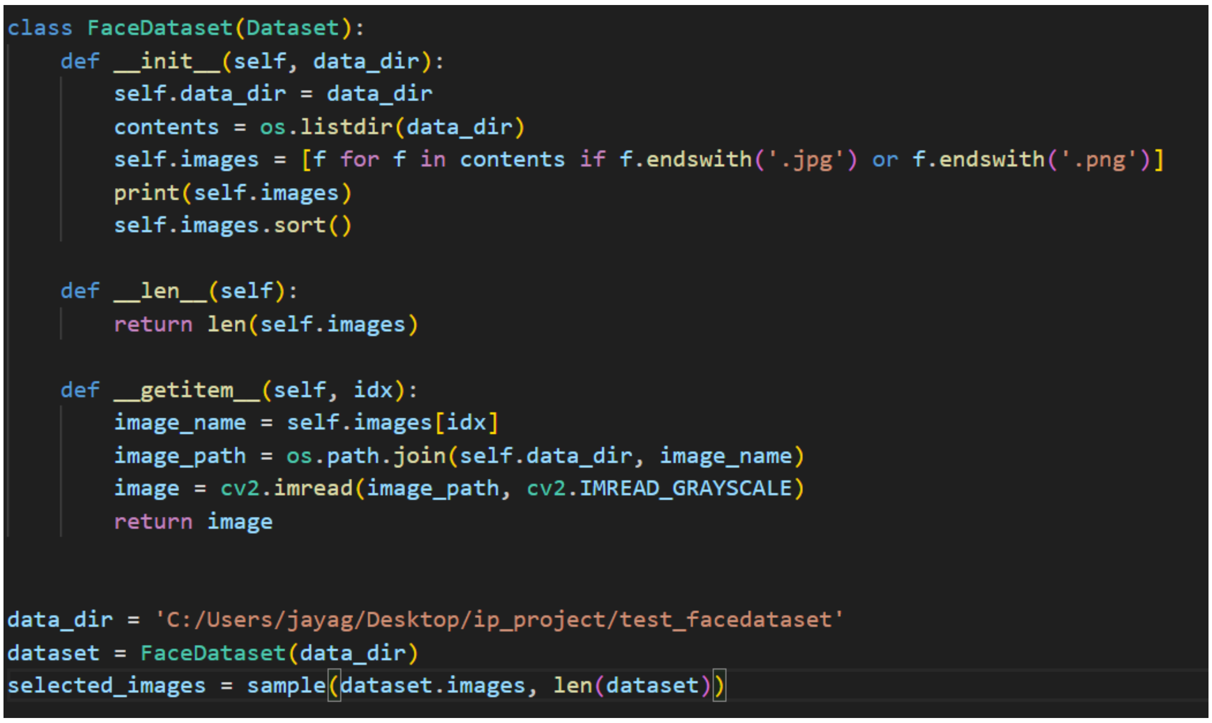
Loading the saved model



```
# Load the saved Random Forest model
model_filename = 'random_forest_model.joblib'
loaded_model = load(model_filename)
```

This segment of code loads the previously trained Random Forest model from the file named 'random_forest_model.joblib' using the *load()* function from the joblib module.

Loading test_facedataset

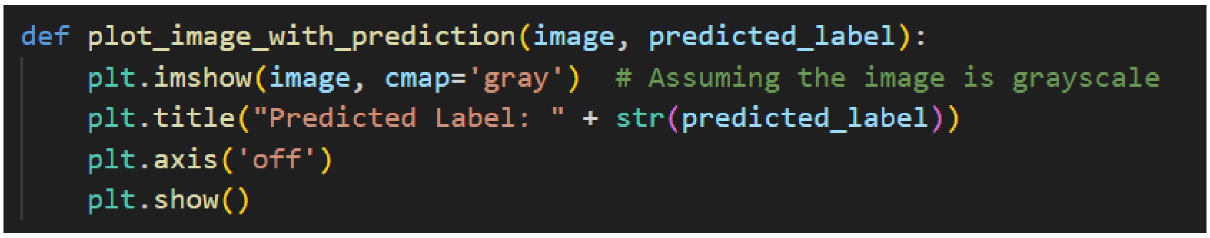


```
class FaceDataset(Dataset):
    def __init__(self, data_dir):
        self.data_dir = data_dir
        contents = os.listdir(data_dir)
        self.images = [f for f in contents if f.endswith('.jpg') or f.endswith('.png')]
        print(self.images)
        self.images.sort()

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        image_name = self.images[idx]
        image_path = os.path.join(self.data_dir, image_name)
        image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        return image


data_dir = 'C:/Users/jayag/Desktop/ip_project/test_facedataset'
dataset = FaceDataset(data_dir)
selected_images = sample(dataset.images, len(dataset))
```

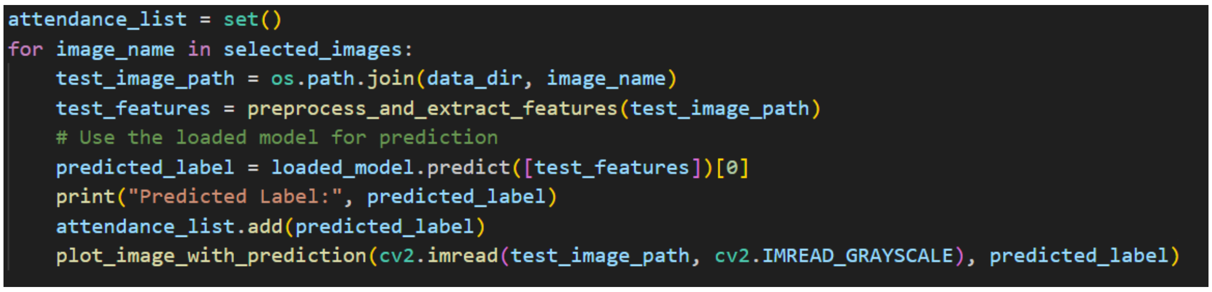
The FaceDataset class encapsulates a dataset of face images, facilitating data handling for machine learning tasks. Upon instantiation with a directory path containing face images, it organizes the data by listing and filtering image files based on their extensions (.jpg or .png).

With its __len__ method, the class provides the total number of images in the dataset. Moreover, the __getitem__ method retrieves individual items from the dataset, providing access to images at specific indices.

Model Testing and visualization



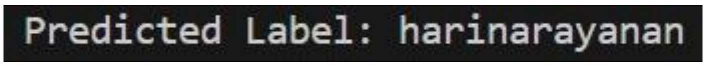
```python
def plot_image_with_prediction(image, predicted_label):
    plt.imshow(image, cmap='gray')  # Assuming the image is grayscale
    plt.title("Predicted Label: " + str(predicted_label))
    plt.axis('off')
    plt.show()
```



```python
attendance_list = set()
for image_name in selected_images:
    test_image_path = os.path.join(data_dir, image_name)
    test_features = preprocess_and_extract_features(test_image_path)
    # Use the loaded model for prediction
    predicted_label = loaded_model.predict([test_features])[0]
    print("Predicted Label:", predicted_label)
    attendance_list.add(predicted_label)
    plot_image_with_prediction(cv2.imread(test_image_path, cv2.IMREAD_GRAYSCALE), predicted_label)
```

In the provided code segment, each selected image from the list `selected_images` is processed iteratively. For each image, its full path is generated by combining the directory path (`data_dir`) with the image name. Features are then extracted from the image using the *preprocess_and_extract_features()* function.

The loaded Random Forest model is employed to predict the label corresponding to the extracted features. The predicted label is printed to the console for visibility. Additionally, the predicted label is added to the `attendance_list` set, ensuring attendance tracking.



```
Predicted Label: harinarayanan
```

Figure 3: Output on console

To visualize the prediction outcome, the *plot_image_with_prediction()* function is called, displaying the test image alongside its predicted label in grayscale. This facilitates manual verification of the model's performance by providing a visual confirmation of the predicted labels for each image.

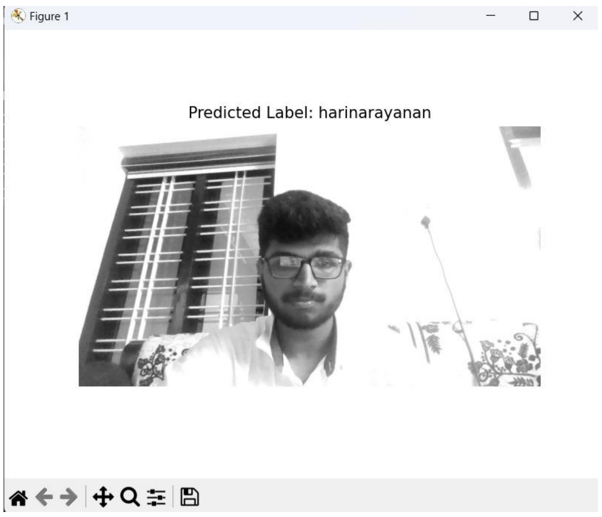
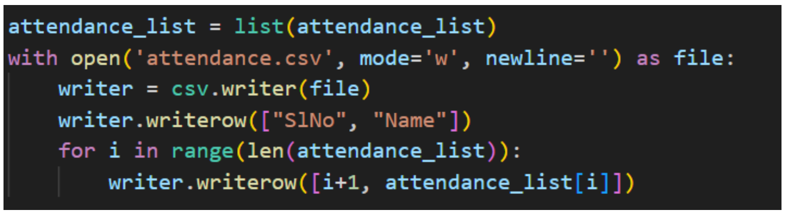


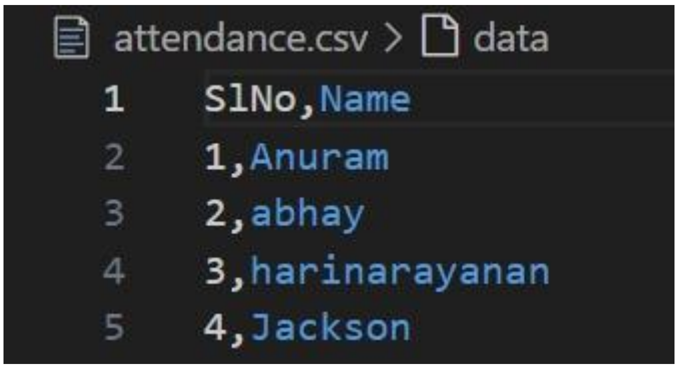
Figure 4: Predicted label on the image

Saving the results



```
attendance_list = list(attendance_list)
with open('attendance.csv', mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["SlNo", "Name"])
    for i in range(len(attendance_list)):
        writer.writerow([i+1, attendance_list[i]])
```

A csv file titled "attendance.csv" is opened in write mode. The predicted labels are then written into this file, facilitating the tracking of attendance.



```
SlNo,Name
1,Anuram
2,abhay
3,harinarayanan
4,Jackson
```

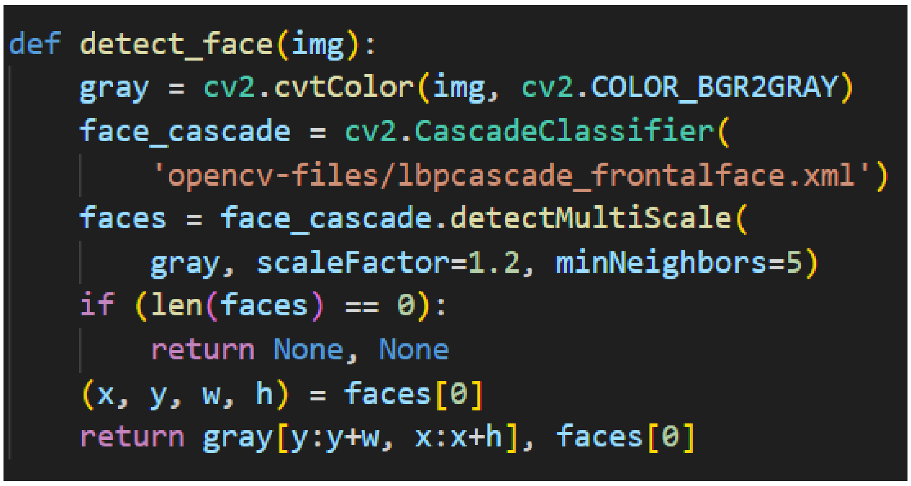
Figure 5: attendance.csv

LBPH

File 3: detect.py

Face detection

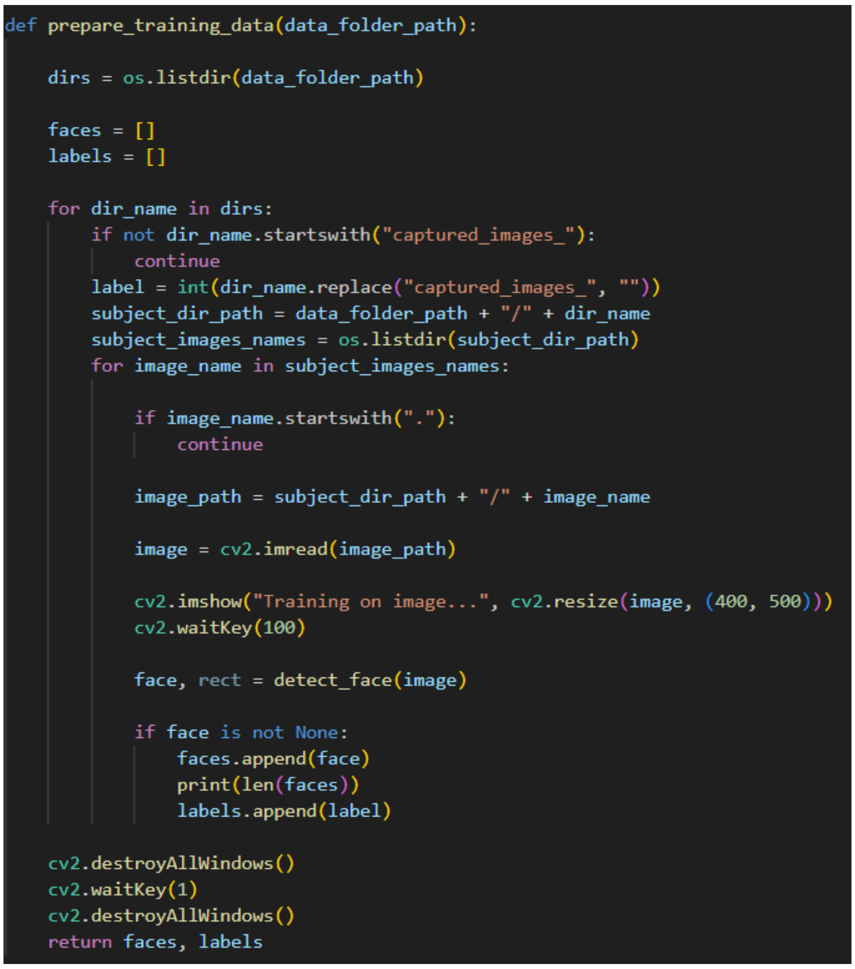
Function for face detection within an image. Since the face detection algorithm works on grayscale images, the input image is converted to a grayscale image. The haar cascade classifier is loaded for frontal face detection using OpenCV's Cascade Classifier class. The method detectMultiScale detects the faces and returns a list of rectangles representing the detected faces. It selects the first face from the list and returns the ROI along with the bounding box coordinates. If there are no faces detected, it will return 'None, None'.



```
def detect_face(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    face_cascade = cv2.CascadeClassifier(
        'opencv-files/lbpcascade_frontalface.xml')
    faces = face_cascade.detectMultiScale(
        gray, scaleFactor=1.2, minNeighbors=5)
    if (len(faces) == 0):
        return None, None
    (x, y, w, h) = faces[0]
    return gray[y:y+w, x:x+h], faces[0]
```

Training Data

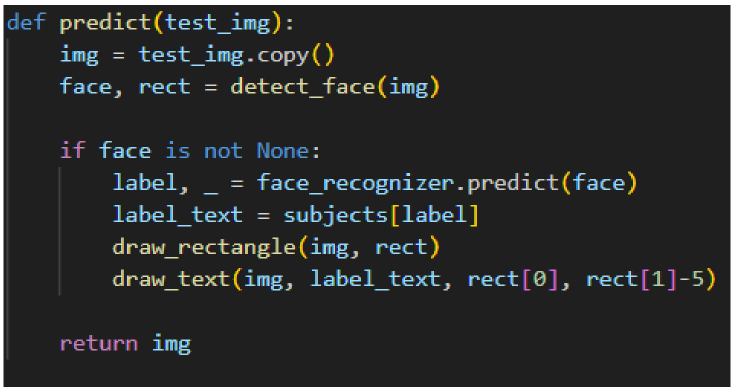
This function is used to prepare the training data. Empty lists for faces and labels are initialized. For each image in the dataset the detect_face function is called and the detected face is appended to the faces list and corresponding label in the labels list. The function returns the list of detected faces and corresponding labels.



```python
def prepare_training_data(data_folder_path):

    dirs = os.listdir(data_folder_path)

    faces = []
    labels = []

    for dir_name in dirs:
        if not dir_name.startswith("captured_images_"):
            continue
        label = int(dir_name.replace("captured_images_", ""))
        subject_dir_path = data_folder_path + "/" + dir_name
        subject_images_names = os.listdir(subject_dir_path)
        for image_name in subject_images_names:

            if image_name.startswith("."):
                continue

            image_path = subject_dir_path + "/" + image_name

            image = cv2.imread(image_path)

            cv2.imshow("Training on image...", cv2.resize(image, (400, 500)))
            cv2.waitKey(100)

            face, rect = detect_face(image)

            if face is not None:
                faces.append(face)
                print(len(faces))
                labels.append(label)

    cv2.destroyAllWindows()
    cv2.waitKey(1)
    cv2.destroyAllWindows()
    return faces, labels
```

Face Recognition

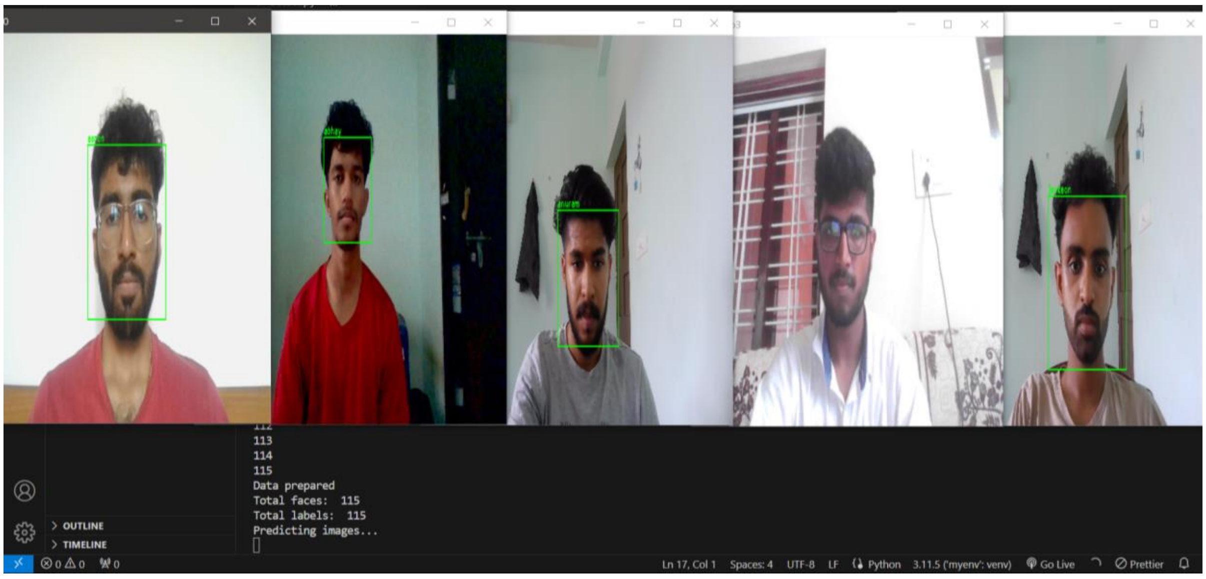
This function is used to predict the identity of an input image. The detect_face method is invoked to detect faces in the image. The label of the detected face is predicted using the face_recognizer method and assigns the label to it. A rectangle is drawn around the detected face and the name of the subject is retrieved using the label predicted. The modified image is finally returned.



```python
def predict(test_img):
    img = test_img.copy()
    face, rect = detect_face(img)

    if face is not None:
        label, _ = face_recognizer.predict(face)
        label_text = subjects[label]
        draw_rectangle(img, rect)
        draw_text(img, label_text, rect[0], rect[1]-5)

    return img
```

Visualisation and Results

The detected face can be visualized by the bounding box and the predicted label is displayed above the box.


```
113
114
115
Data prepared
Total faces:  115
Total labels:  115
Predicting images...
```

5.2 Source Code

lbp_hog_train.py

```
import os
import cv2
import numpy as np
from torch.utils.data import Dataset
from skimage.feature import local_binary_pattern, hog
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from random import Random, sample
import matplotlib.pyplot as plt
from joblib import dump

def plot_image_with_prediction(image, predicted_label):
    plt.imshow(image, cmap='gray')
    plt.title("Predicted Label: " + str(predicted_label))
    plt.axis('off')
    plt.show()

def extract_lbp_features(image):
    lbp_radius = 3
    lbp_points = 24
    lbp_image = local_binary_pattern(image, lbp_points, 
lbp_radius, method='uniform')
    hist, _ = np.histogram(lbp_image.ravel(), bins=np.arange(0, 
lbp_points + 3), range=(0, lbp_points + 2))
    hist = hist.astype("float")
    hist /= (hist.sum() + 1e-7)
    return hist
```

```
def extract_hog_features(image):
    hog_features = hog(image, orientations=9, pixels_per_cell=(8,
8),
                          cells_per_block=(2,                  2),
transform_sqrt=True, block_norm="L2-Hys")
    return hog_features

class FaceDataset(Dataset):
    def __init__(self, data_dir):
        self.data_dir = data_dir
        contents = os.listdir(data_dir)
        self.images = [f for f in contents if f.endswith('.jpg')]
        self.images.sort()

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        image_name = self.images[idx]
        label = image_name.split('_')[0]
        image_path = os.path.join(self.data_dir, image_name)
        image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        image.resize(720, 1280)
        # plt.imshow(image, cmap='gray')
        # plt.show()
        return (image, label)

X_features = []
y_labels = []

data_dir = 'C:/Users/jayag/Desktop/ip_project/facedataset'
dataset = FaceDataset(data_dir)

index_list = [i for i in range(len(dataset))]
for _ in range(len(dataset)):
    idx = Random().choice(index_list)
    image, label = dataset[idx]
    lbp_features = extract_lbp_features(image)
    hog_features = extract_hog_features(image)
    combined_features        =        np.concatenate((lbp_features,
hog_features))
    X_features.append(combined_features)
    y_labels.append(label)

X_features = np.array(X_features)
y_labels = np.array(y_labels)
X_train,  X_test,  y_train,  y_test  =  train_test_split(X_features,
y_labels, test_size=0.2, random_state=42)
rf_classifier      =      RandomForestClassifier(n_estimators=100,
random_state=42)
rf_classifier.fit(X_train, y_train)
y_pred = rf_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

model_filename = 'random_forest_model.joblib'
```

```
dump(rf_classifier, model_filename)
```

lbp_hog_test.py

```
from joblib import load
import cv2
import numpy as np
from skimage.feature import local_binary_pattern, hog
import os
from torch.utils.data import Dataset
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from random import sample
import matplotlib.pyplot as plt
import csv

def take_picture():
    cam = cv2.VideoCapture(0)
    cam.set(3, 1280)
    cv2.namedWindow("test")
    img_counter = 0
    while True:
        ret, frame = cam.read()
        if not ret:
            print("failed to grab frame")
            break
        cv2.imshow("test", frame)
        k = cv2.waitKey(1)
        if k%256 == 27:
            print("Escape hit, closing...")
            break
        elif k%256 == 32:
            img_name                                                  =
"C:/Users/jayag/Desktop/ip_project/test_facedataset/face_{}.jpg".f
ormat(img_counter)
            cv2.imwrite(img_name, frame)
            print("{} written!".format(img_name))
            img_counter += 1
    cam.release()
    cv2.destroyAllWindows()

take_picture()

def extract_lbp_features(image):
    lbp_radius = 3
    lbp_points = 24
    lbp_image     =     local_binary_pattern(image,     lbp_points,
lbp_radius, method='uniform')
    hist,  _  =  np.histogram(lbp_image.ravel(),  bins=np.arange(0,
lbp_points + 3), range=(0, lbp_points + 2))
    hist = hist.astype("float")
    hist /= (hist.sum() + 1e-7)
    return hist

def extract_hog_features(image):
    hog_features = hog(image, orientations=9, pixels_per_cell=(8,
8),
                              cells_per_block=(2,                  2),
transform_sqrt=True, block_norm="L2-Hys")
    return hog_features

# Load the saved Random Forest model
model_filename = 'random_forest_model.joblib'
loaded_model = load(model_filename)

def preprocess_and_extract_features(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    image.resize((720, 1280))
    lbp_features = extract_lbp_features(image)
    hog_features = extract_hog_features(image)
    combined_features        =       np.concatenate((lbp_features,
hog_features))
    return combined_features

class FaceDataset(Dataset):
    def __init__(self, data_dir):
        self.data_dir = data_dir
        contents = os.listdir(data_dir)
        self.images = [f for f in contents if f.endswith('.jpg') ]
        print(self.images)
        self.images.sort()

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        image_name = self.images[idx]
        image_path = os.path.join(self.data_dir, image_name)
        image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        return image

def plot_image_with_prediction(image, predicted_label):
    plt.imshow(image,  cmap='gray')    #  Assuming  the  image  is
grayscale
    plt.title("Predicted Label: " + str(predicted_label))
    plt.axis('off')
    plt.show()

data_dir = 'C:/Users/jayag/Desktop/ip_project/test_facedataset'
dataset = FaceDataset(data_dir)
selected_images = sample(dataset.images, len(dataset))

attendance_list = set()
for image_name in selected_images:
    test_image_path = os.path.join(data_dir, image_name)
    test_features                                                  =
preprocess_and_extract_features(test_image_path)
    # Use the loaded model for prediction
    predicted_label = loaded_model.predict([test_features])[0]
    print("Predicted Label:", predicted_label)
    attendance_list.add(predicted_label)
    plot_image_with_prediction(cv2.imread(test_image_path,
cv2.IMREAD_GRAYSCALE), predicted_label)

attendance_list = list(attendance_list)
with open('attendance.csv', mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["SlNo", "Name"])
    for i in range(len(attendance_list)):
        writer.writerow([i+1, attendance_list[i]])
```

References

[1] Agarwal, Mayank & Jain, Nikunj & Kumar, Manish & Agrawal, Himanshu. (2010). Face Recognition Using Eigen Faces and Artificial Neural Network. International Journal of Computer Theory and Engineering. 624-629. 10.7763/IJCTE.2010.V2.213.

[2] Mady, Huda & Hilles, Shadi M. (2018). Face recognition and detection using Random Forest and combination of LBP and HOG features. 1-7. 10.1109/ICSCEE.2018.8538377.

[3] Farah Deeba, Hira Memon, Fayaz Ali Dharejo, Aftab Ahmed and Abddul Ghaffar, “LBPH-based Enhanced Real-Time Face Recognition” International Journal of Advanced Computer Science and Applications(IJACSA), 10(5), 2019.