CONTENTS

INTRODUCTION

Facial recognition technology has revolutionized the way we interact with security systems, digital devices, and even social media platforms. By combining the power of machine learning algorithms and computer vision techniques, facial recognition systems can identify and verify individuals based on their unique facial features.

At the heart of facial recognition technology lies the field of machine learning, which enables computers to learn from data and make predictions or decisions without being explicitly programmed. Machine learning algorithms analyze vast amounts of facial data to extract meaningful patterns and features that distinguish one face from another. These algorithms continuously improve their performance as they are exposed to more data, making them increasingly accurate over time.

OpenCV (Open-Source Computer Vision Library) is a widely used open-source library that provides tools and functions for real-time computer vision applications. It offers a plethora of functionalities for image and video processing, including facial detection, recognition, and tracking. Combining OpenCV with machine learning algorithms creates a powerful framework for building robust facial recognition systems.

In this context, facial recognition systems typically involve several key steps:

1. **Face Detection**: The first step is to locate, and extract faces from images or video streams. OpenCV provides pre-trained models and functions for detecting faces in real-time.

2. **Feature Extraction**: Once faces are detected, relevant features such as the distance between eyes, nose shape, and jawline are extracted. These features are crucial for accurately representing each face in a format suitable for machine learning algorithms.

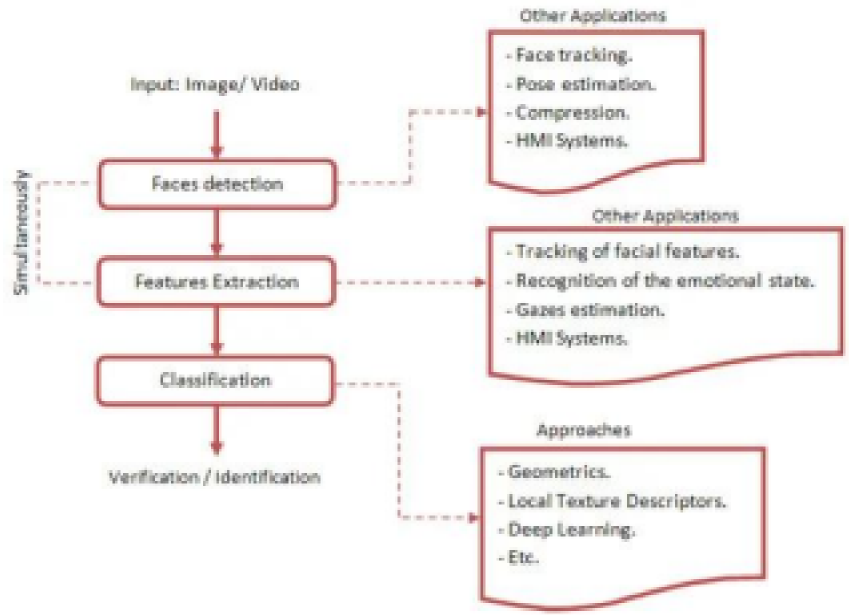
3. **Training a Classifier**: Machine learning algorithms, such as Support Vector Machines (SVM), Convolutional Neural Networks (CNN), or deep learning architectures like Siamese networks, are trained on labeled facial data. During training, the algorithm learns to differentiate between different individuals based on their facial features.

4. **Face Recognition**: After training, the classifier can recognize faces by comparing the extracted features of a new face with those stored in its database. The algorithm assigns a label or identity to the face based on its closest match in the database.

5. **Verification or Identification**: Facial recognition systems can be used for both verification (one-to-one matching) and identification (one-to-many matching). In verification, the system verifies if the presented face matches the claimed identity, while in identification, it searches a database to identify the person in the image.

Facial recognition using machine learning algorithms and OpenCV has numerous applications across various domains, including security, access control, surveillance, personalization, and more. However, it also raises important ethical and privacy considerations regarding data protection, consent, and potential biases in the algorithms. As technology continues to advance, it is essential to strike a balance between innovation and responsible use to ensure its beneficial and ethical deployment in society.

BLOCK DIAGRAM:



1. Face detection is the first step in the automated face recognition system. It usually determines whether an image includes a face(s). If it does, its function is to trace one or several face locations in the picture
2. Feature extraction step consists of extracting from the detected face a feature vector named the signature, which must be enough to represent a face. The individuality of the face and the property of distinguishing between two separate persons must be checked
3. Classification involves verification and identification. Verification requires matching one face to another to authorize access to a requested identity. Identification compares a face to several other faces that are given with several possibilities to find the face's identity.

CLASSIFICATION PROBLEM:

For a facial recognition system, the problem typically involves classifying images of faces into different categories, such as identifying specific individuals or recognizing facial expressions. Here's how we can approach building a facial recognition system:

Data Collection: Gather a large dataset of facial images. Ensure that the dataset covers a diverse range of identities, poses, expressions, lighting conditions, and backgrounds to make the model robust.

Data Preprocessing: Preprocess the facial images to ensure they are aligned, normalized, and of consistent size. Common preprocessing steps include face detection, cropping, resizing, and normalization.

Model Selection: Choose an appropriate deep learning architecture for your facial recognition task. Convolutional Neural Networks (CNNs) are commonly used for image classification tasks due to their ability to capture spatial hierarchies of features. Few other classifiers can be RandomForestClassifier and HaarCascadeClassifier.

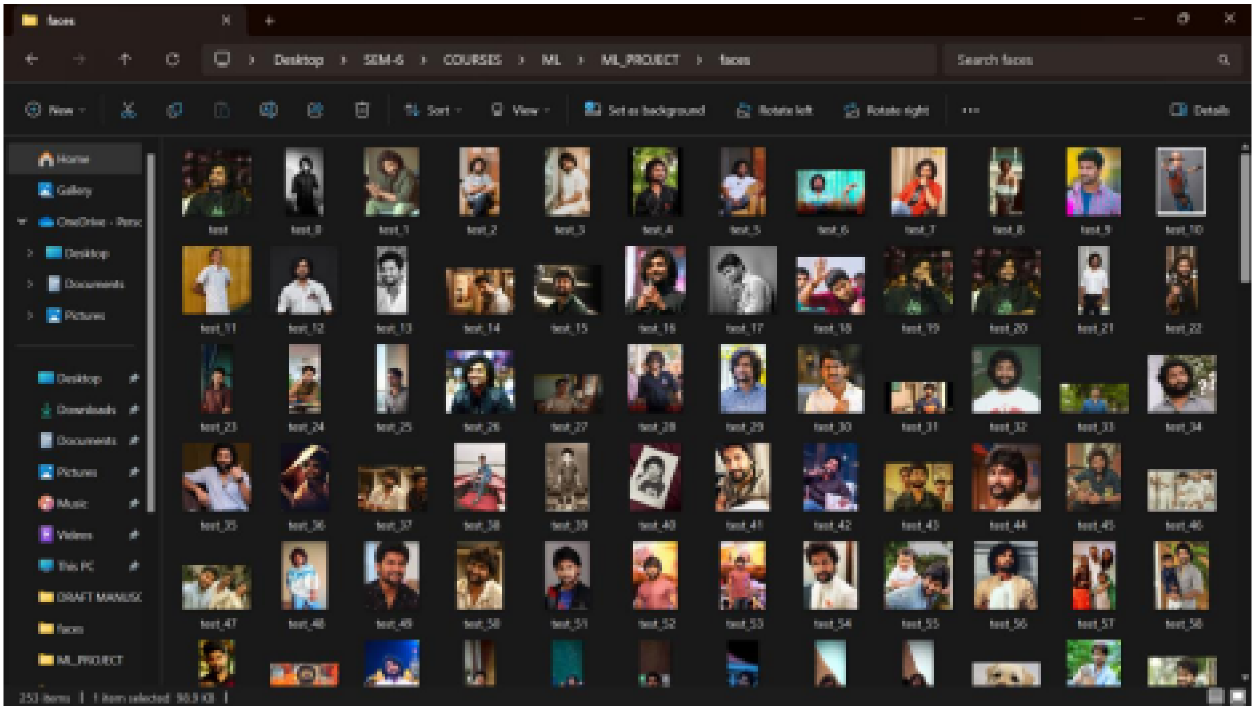
Model Training: Split your dataset into training, validation, and test sets. Train your chosen model on the training data, using the validation set to tune hyperparameters and monitor the model's performance.

Evaluation: Evaluate the trained model on the test set to assess its performance. Metrics such as accuracy, precision, recall, and F1-score can be used to evaluate classification performance.

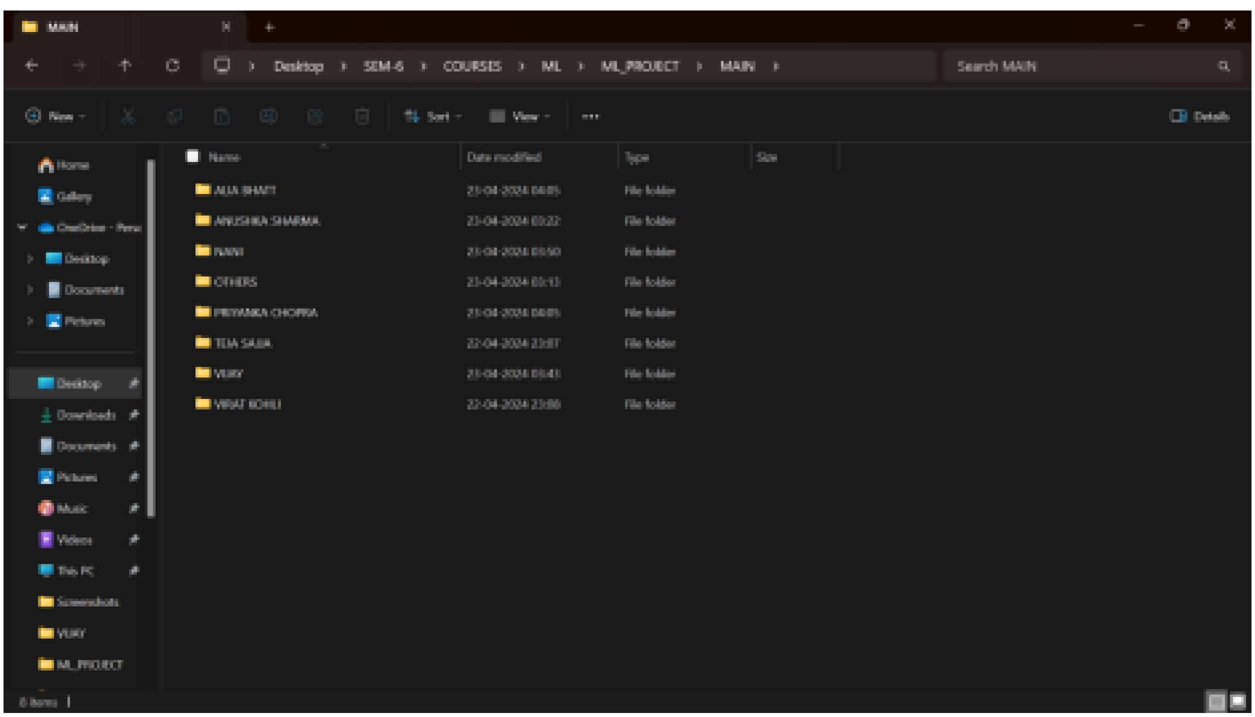
DATASET CREATION:

The data set is created in two ways

- The data set is created as an Unlabeled Dataset for the Face Identification model for the Conventional Facial Recognition Algorithm.
- The data set is created as a Labelled Dataset for the prediction done using RandomForestClassifier and using a CNN model.
    - The description of the dataset is as follows:



The dataset consists of a total of 401 images which are a collection of different classes of people.



Above is a detailed description of the different classes of images of people taken and class wise samples are shown below:

NANI : 65 Images

ALIA BHATT : 79 Images

PRIYANKA CHOPRA : 69 Images

ANUSHKA SHARMA: 68 Images

VIJAY : 51 Images

TEJA SAJJA : 15 Images

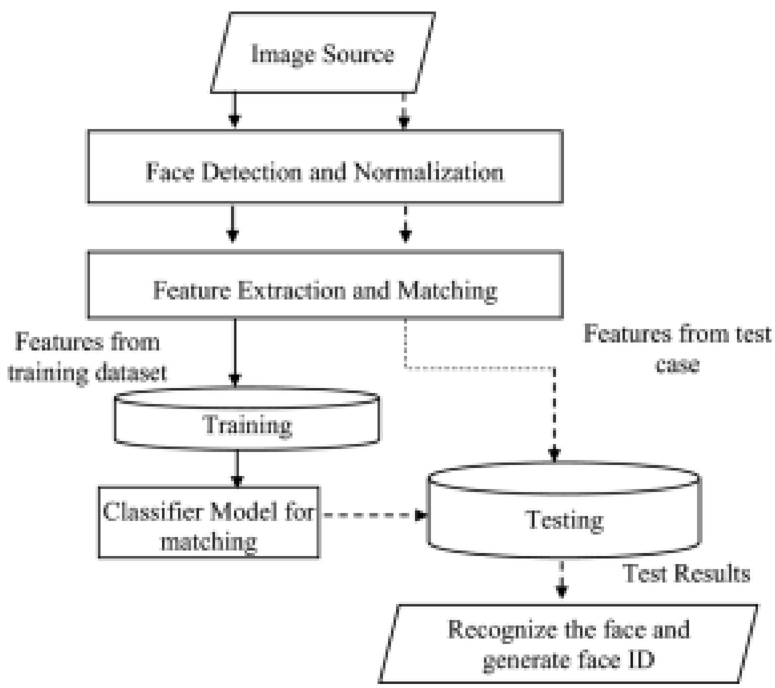
OTHERS : 5 Images

VIRAT KOHLI : 49 Images

METHODOLOGY:

The basic block diagram for the facial recognition system is shown

below



Now that we have been introduced with the flow of the facial recognition system, let us now dive deep into how each step works
In Machine Learning, Facial Recognition is done using OpenCV. OpenCV is the world's biggest computer vision library, offering over 2500 algorithms and tools for image and video manipulation, object and face detection, deep learning, and more.

The first step involves Feature Extraction. This is the main step under facial recognition. This step involves extracting various types of features from the image which are referred to as "Face Encodings" using which the facial recognition works. Some of the key features which are extracted includes:

Geometric features: These include the distance between the eyes, the width of the nose, the shape of the jawline, and other spatial relationships between facial landmarks.

Texture features: These features capture the texture of various regions of the face, such as the smoothness of the skin, the presence of wrinkles, and the distribution of pores.

Appearance features: These features encompass the overall appearance of the face, including attributes such as hair color, skin tone, presence of glasses or facial hair, and other distinguishing characteristics.

Facial landmarks**:** Points on the face such as the corners of the eyes, the tip of the nose, and the corners of the mouth are detected and used to characterize the facial structure.

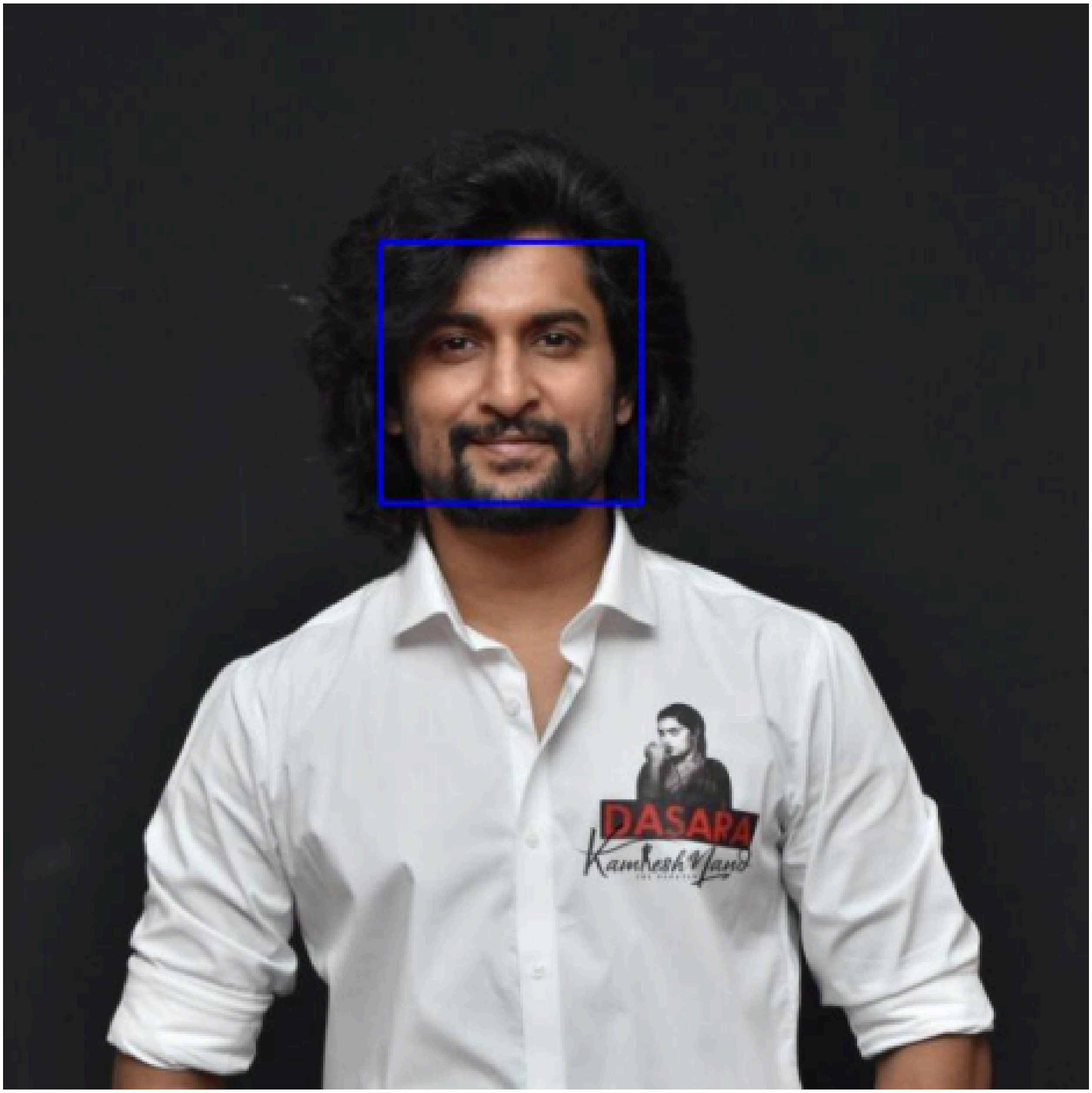
Eigenfaces: These are a set of eigenvectors derived from the covariance matrix of the probability distribution of pixel intensities of facial images. They represent the principal components of variation in a set of face images.

Here's how Haar Cascade typically fits into the process:

Face Detection: Before facial recognition can take place, the system needs to locate faces within an image or video frame. Haar Cascade classifiers are trained to detect the presence of objects with specific features, such as faces. These classifiers use a set of pre-defined features (Haar-like features) to identify regions of interest that may contain faces.

Region of Interest (ROI) Extraction: Once faces are detected by the Haar Cascade classifier, the corresponding regions of interest (ROIs) containing the faces can be extracted from the image. This step is crucial for isolating the facial regions from the rest of the image data, reducing the computational load for subsequent processing steps.

Feature Extraction: After the ROIs are obtained, the facial recognition system can proceed to extract features from these regions. This typically involves techniques such as the ones mentioned earlier (geometric features, texture features, etc.), which are applied specifically to the facial regions identified by the Haar Cascade detector.



STEP 1:

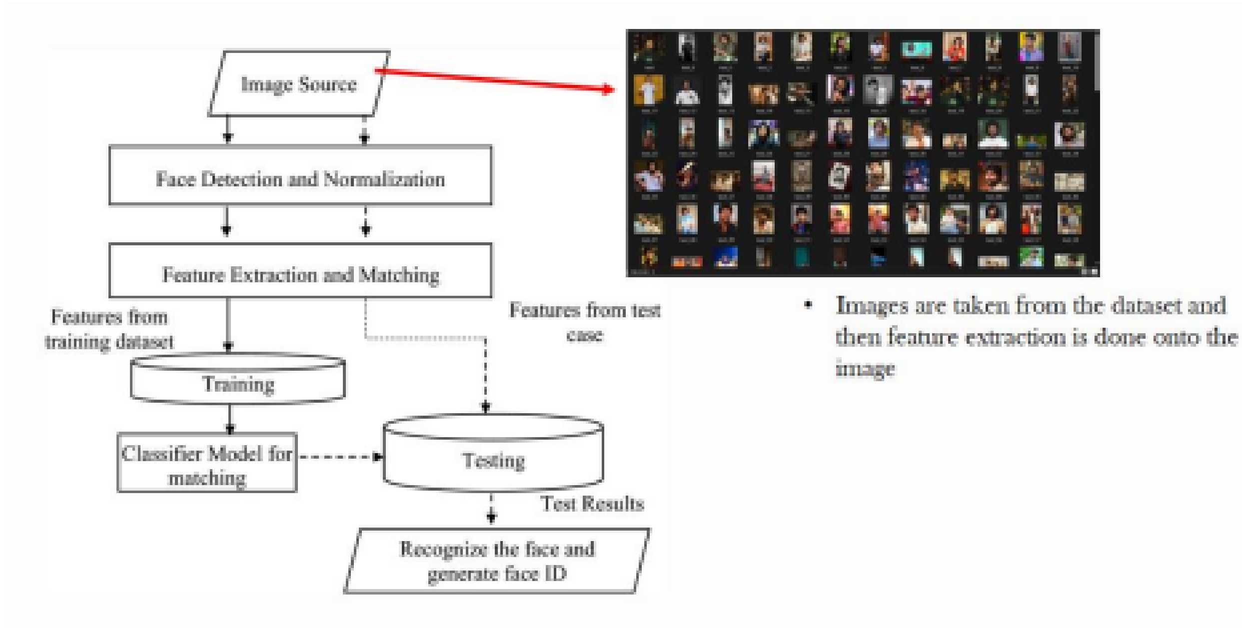


Image Source
Face Detection and Normalization
Feature Extraction and Matching
Features from training dataset
Features from test case
Training
Classifier Model for matching
Testing
Test Results
Recognize the face and generate face ID
• Images are taken from the dataset and then feature extraction is done onto the image

STEP 2:

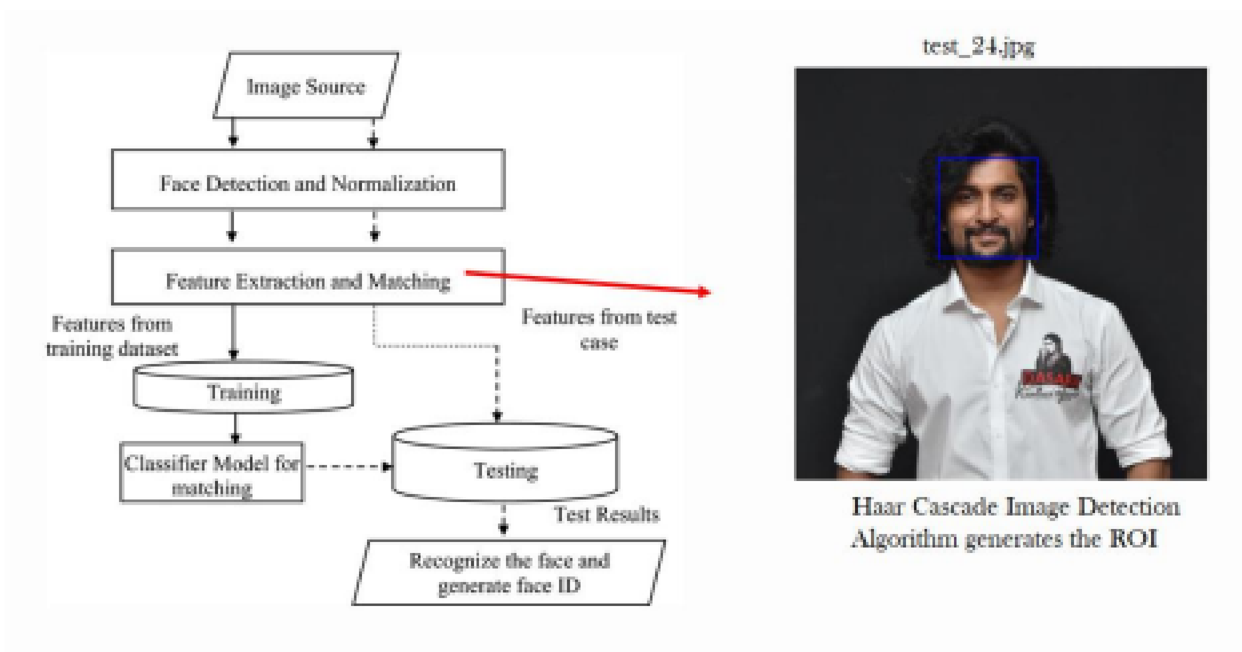


Image Source
Face Detection and Normalization
Feature Extraction and Matching
Features from training dataset
Features from test case
Training
Classifier Model for matching
Testing
Test Results
Recognize the face and generate face ID
test_24.jpg
Haar Cascade Image Detection Algorithm generates the ROI

STEP 3:

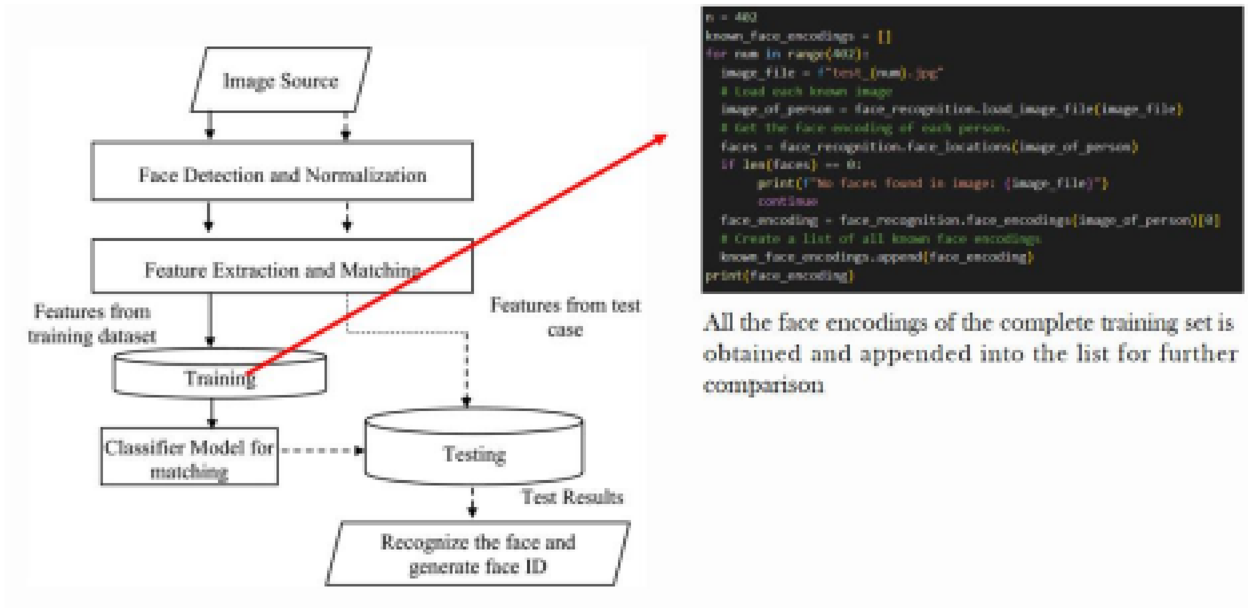


Image Source
Face Detection and Normalization
Feature Extraction and Matching
Features from training dataset
Features from test case
Training
Classifier Model for matching
Testing
Test Results
Recognize the face and generate face ID
n = 402
known_face_encodings = []
for num in range(402):
image_file = f"test_{num}.jpg"
# Load each known image
image_of_person = face_recognition.load_image_file(image_file)
# Get the face encoding of each person.
faces = face_recognition.face_locations(image_of_person)
if len(faces) == 0:
print(f"No faces found in image: {image_file}")
continue
face_encoding = face_recognition.face_encodings(image_of_person)[0]
# Create a list of all known face encodings
known_face_encodings.append(face_encoding)
print(face_encoding)
All the face encodings of the complete training set is obtained and appended into the list for further comparison

STEP 4:

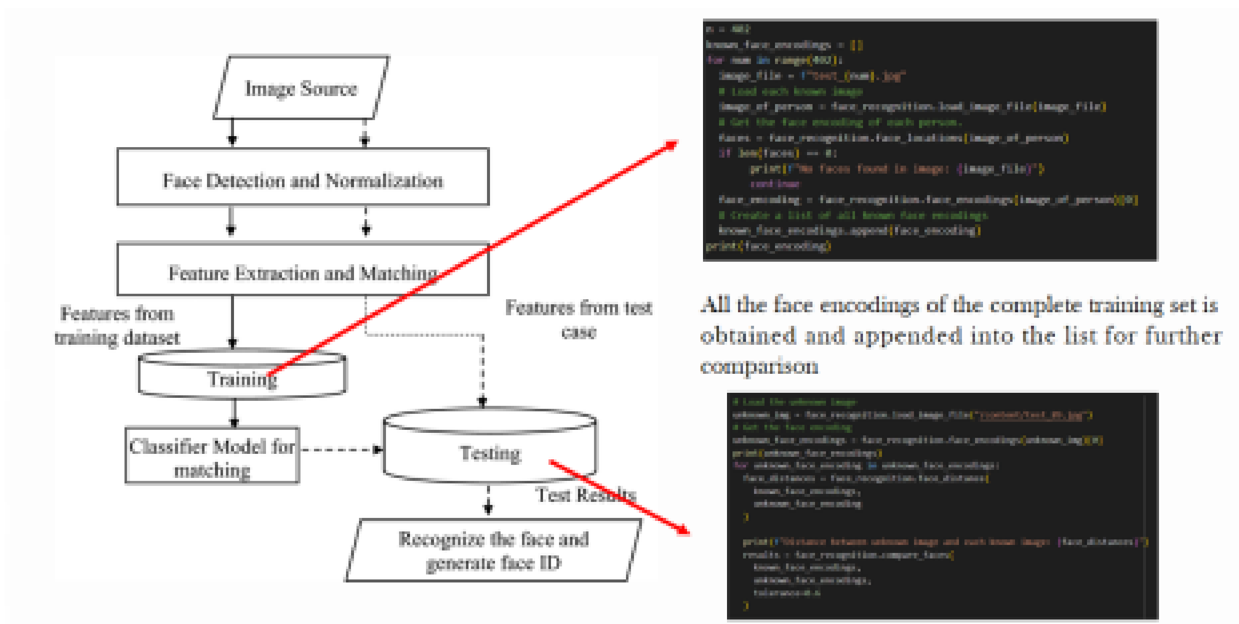
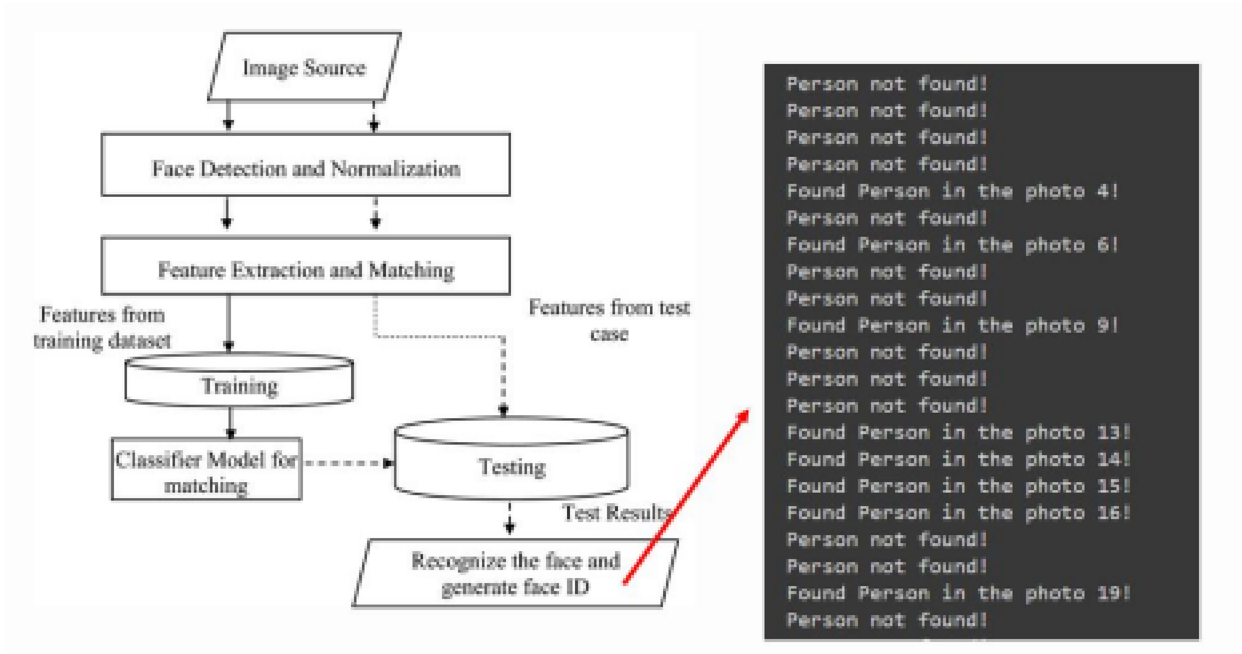


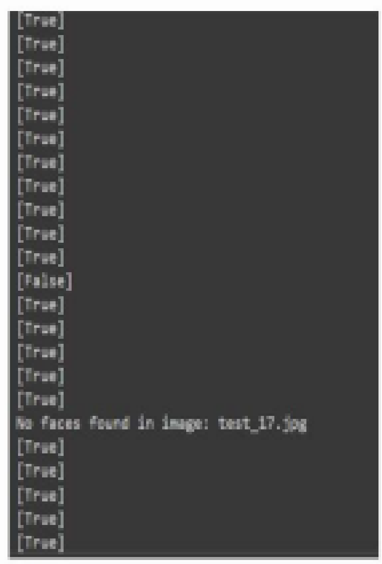
Image Source
Face Detection and Normalization
Feature Extraction and Matching
Features from training dataset
Features from test case
Training
Classifier Model for matching
Testing
Test Results
Recognize the face and generate face ID
All the face encodings of the complete training set is obtained and appended into the list for further comparison

STEP 5:



METHODOLOGY (MANUAL PREDICTION)

- Finding Manual Accuracy when a test image is given as an input.

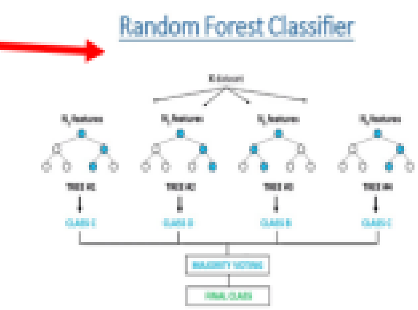


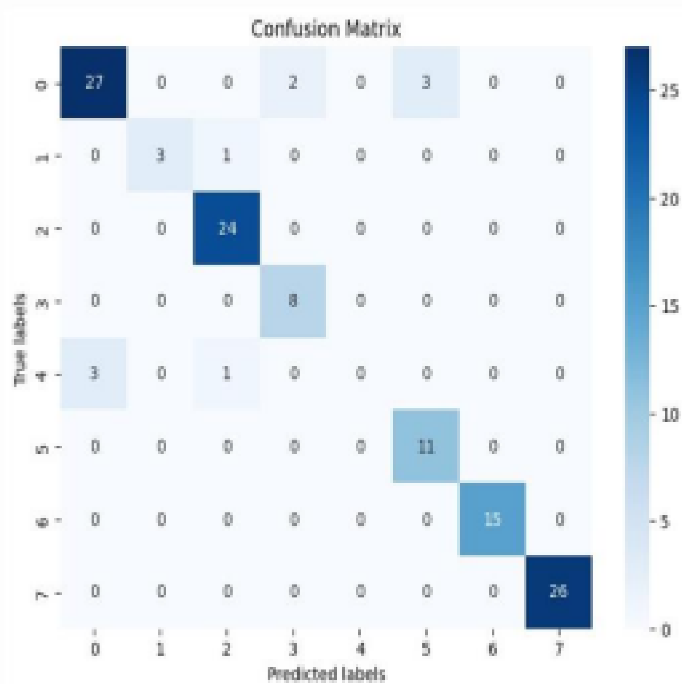
METHODOLOGY (USING RANDOMFOREST CLASSIFIER)

```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)

# Initialize and train the classifier (Random Forest as an example)
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)
print(y_train)
print(y_pred)
print(y_test)
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
matrix = classification_report(y_test,y_pred)
print("Classification Report")
print(matrix)
# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```





Accuracy: 0.92
Classification Report

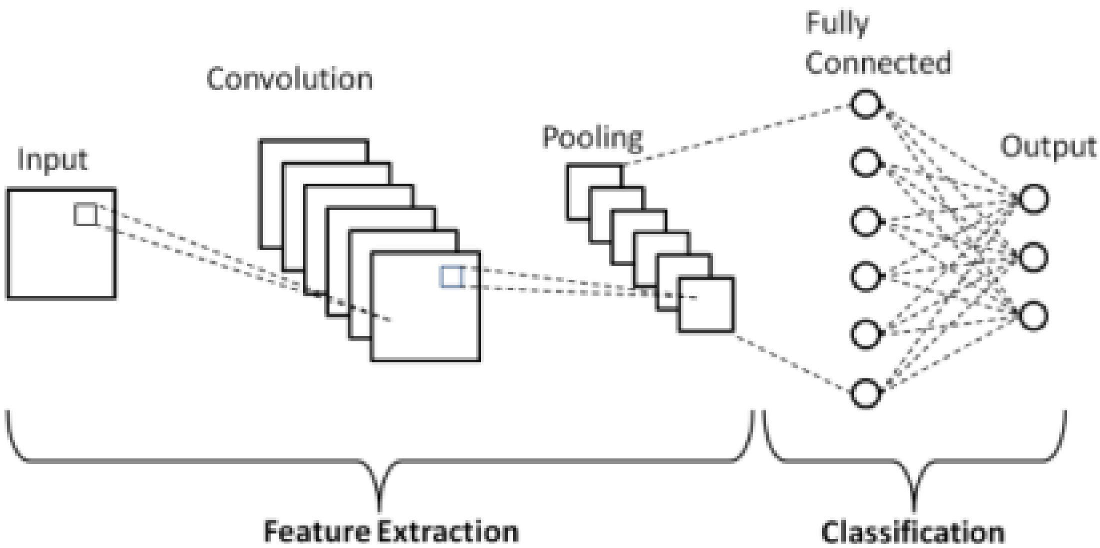
	precision	recall	f1-score	support
Person 0	0.90	0.84	0.87	32
Person 10	1.00	0.75	0.86	4
Person 11	0.92	1.00	0.96	24
Person 4	0.80	1.00	0.89	8
Person 5	0.00	0.00	0.00	4
Person 7	0.79	1.00	0.88	11
Person 8	1.00	1.00	1.00	15
Person 9	1.00	1.00	1.00	26
accuracy			0.92	124
macro avg	0.80	0.82	0.81	124
weighted avg	0.90	0.92	0.90	124

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

COMPARISON WITH CNN MODEL:

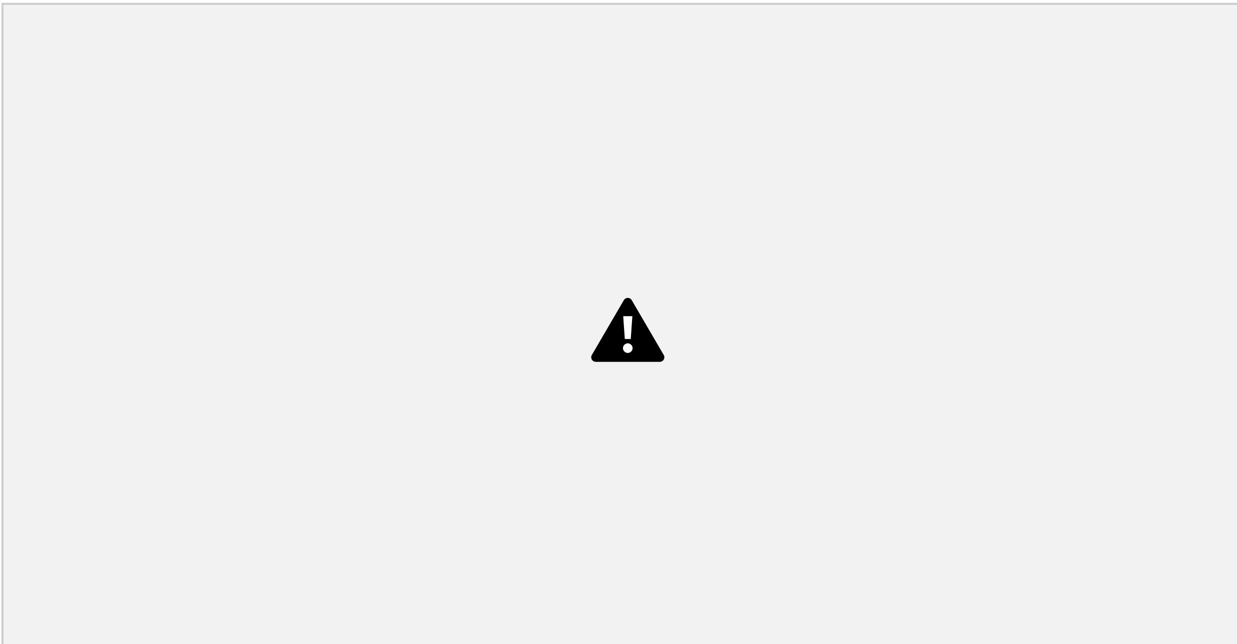


BLOCK DIAGRAM OF CNN MODEL

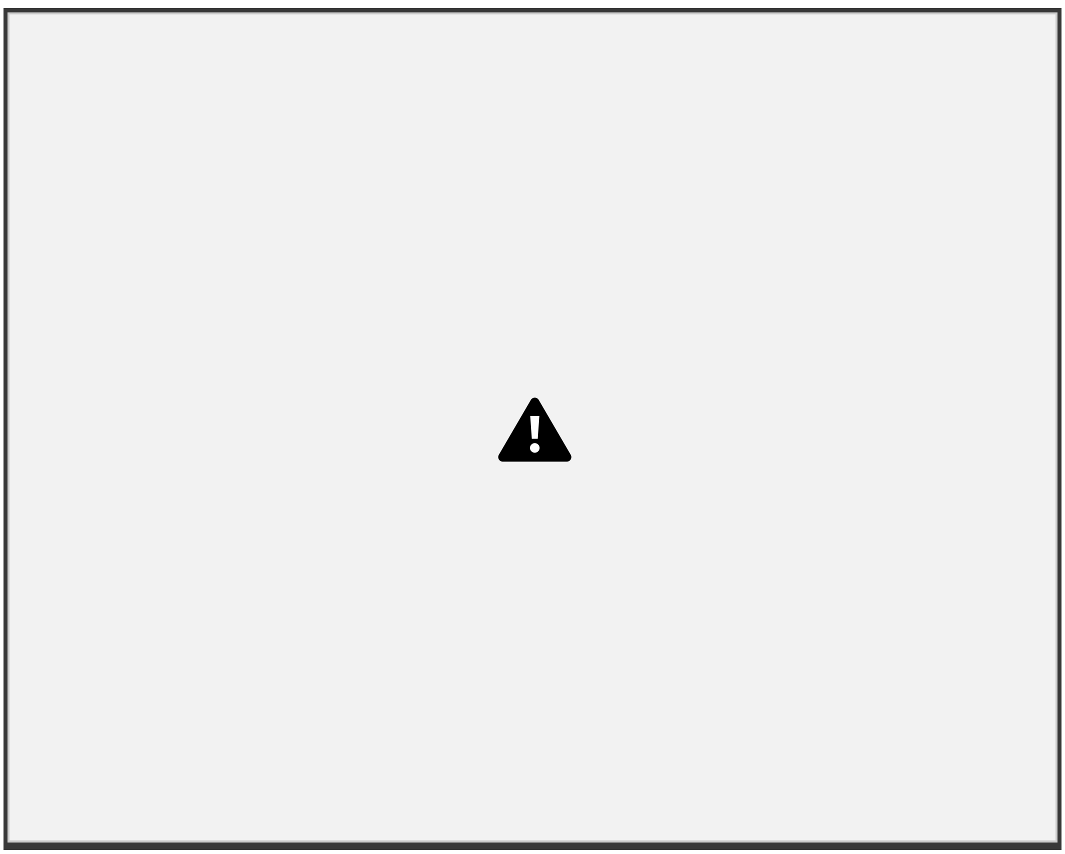
Layer (type)	Output Shape	Param #
conv2d_31 (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d_31 (MaxPooling2D)	(None, 111, 111, 32)	0
batch_normalization_30 (BatchNormalization)	(None, 111, 111, 32)	128
conv2d_32 (Conv2D)	(None, 109, 109, 64)	18496
max_pooling2d_32 (MaxPooling2D)	(None, 54, 54, 64)	0
batch_normalization_31 (BatchNormalization)	(None, 54, 54, 64)	256
conv2d_33 (Conv2D)	(None, 52, 52, 64)	36928
max_pooling2d_33 (MaxPooling2D)	(None, 26, 26, 64)	0
batch_normalization_32 (BatchNormalization)	(None, 26, 26, 64)	256
conv2d_34 (Conv2D)	(None, 24, 24, 96)	55392
max_pooling2d_34 (MaxPooling2D)	(None, 12, 12, 96)	0
batch_normalization_33 (BatchNormalization)	(None, 12, 12, 96)	384
conv2d_35 (Conv2D)	(None, 10, 10, 32)	27680
max_pooling2d_35 (MaxPooling2D)	(None, 5, 5, 32)	0
batch_normalization_34 (BatchNormalization)	(None, 5, 5, 32)	128
dropout_6 (Dropout)	(None, 5, 5, 32)	0
flatten_6 (Flatten)	(None, 800)	0
dense_12 (Dense)	(None, 128)	102528
dense_13 (Dense)	(None, 8)	1032

Total params: 244104 (953.53 KB)
Trainable params: 243528 (951.28 KB)
Non-trainable params: 576 (2.25 KB)

TRAINING THE MODEL WITH THE DATASET

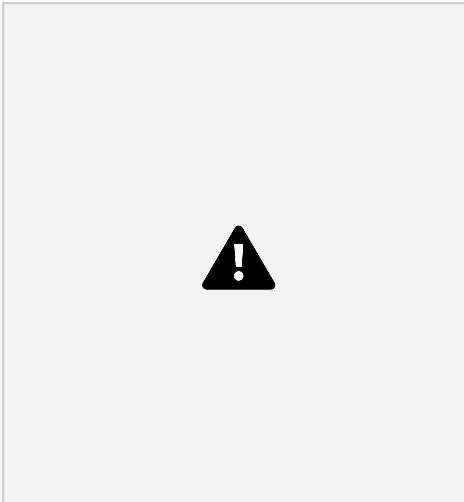


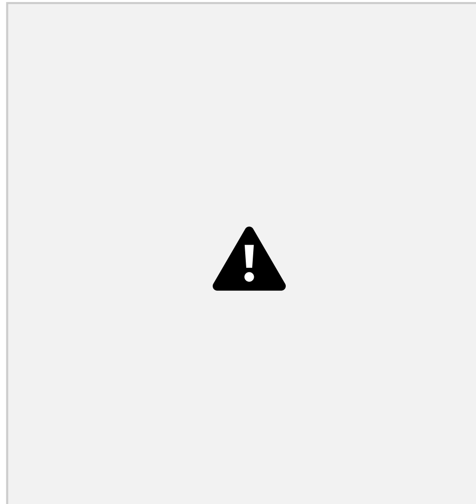
ACCURACY V/S LOSS

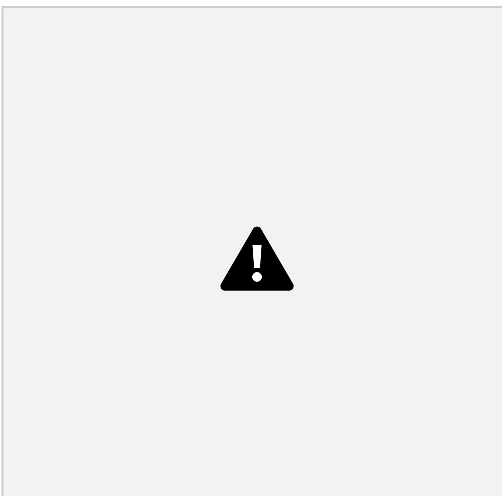


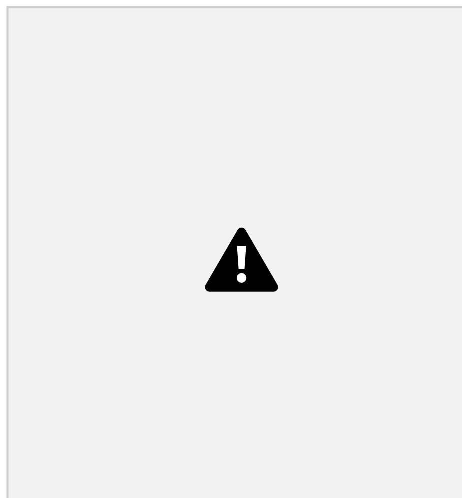
DIFFERENT TEST CASES FOR CLASSIFYING USING THE TRAINED CNN

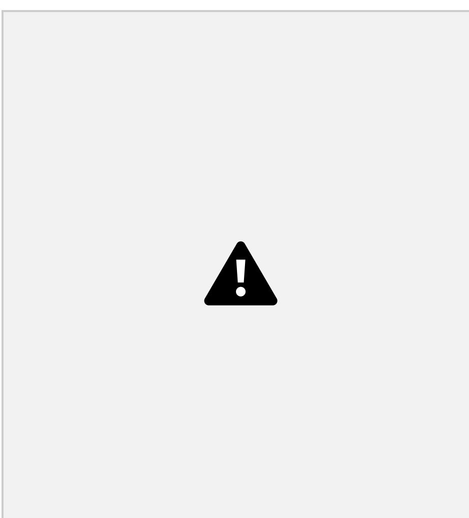
MODEL:











CODE:

```
!pip install opencv-python
```

```
!pip install face-recognition
!pip install cmake
!pip install face_recognition
!pip install dlib
!pip install numpy

from google.colab.patches import cv2_imshow
import cv2
import matplotlib.pyplot as plt
import face_recognition


image=cv2.imread('/content/test_12.jpg')
image
img_1=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
plt.imshow(img_1)
import cv2
from google.colab.patches import cv2_imshow

face_cascade=cv2.CascadeClassifier('haarcascade_frontalface_default.xml'
)  img=cv2.imread('/content/test_12.jpg')
gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
faces=face_cascade.detectMultiScale(gray,1.1,4)
for(x,y,w,h) in faces:
  cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
cv2_imshow(img)
cv2.waitKey()
img_1 =
face_recognition.load_image_file("/content/test_12.jpg")
face_encoding = face_recognition.face_encodings(img_1)[0]
print(face_encoding)
n = 402
known_face_encodings = []
for num in range(402):
  image_file = f"test_{num}.jpg"
  # Load each known image
  image_of_person =
  face_recognition.load_image_file(image_file)  # Get the face
  encoding of each person.
  faces =
  face_recognition.face_locations(image_of_person)  if
  len(faces) == 0:
      print(f"No faces found in image: {image_file}")
```

```
            continue
    face_encoding = face_recognition.face_encodings(image_of_person)[0]
    # Create a list of all known face encodings
    known_face_encodings.append(face_encoding)
print(face_encoding)

# Load the unknown image
unknown_img =
face_recognition.load_image_file("/content/test_89.jpg") # Get the
face encoding
unknown_face_encodings =
face_recognition.face_encodings(unknown_img)[0]
print(unknown_face_encodings)
for unknown_face_encoding in unknown_face_encodings:
    face_distances = face_recognition.face_distance(
        known_face_encodings,
        unknown_face_encoding
    )

    print(f"Distance between unknown image and each known image:
{face_distances}")
    results = face_recognition.compare_faces(
        known_face_encodings,
        unknown_face_encodings,
        tolerance=0.6
    )
for num in range(397):
        if results [num]:
            print(f"Found Person in the photo {num}!")
        else:
            print("Person not found!")
```

TESTING FOR THE ACCURACY USING RANDOM FORST CLASSIFIER:

```
from sklearn.metrics import accuracy_score,
confusion_matrix from sklearn.model_selection import
train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn import metrics

# Ground truth dictionary mapping filenames to person
IDs ground_truth = {}
```

```
for i in range(402):
    filename = f"test_{i}.jpg"
    if i == 38:
        person = "Person 0"
    elif i == 39:
        person = "Person 0"
    elif i == 68:
        person = "Person 0"
    elif i == 11:
        person = "Person 0"
    elif i == 57:
        person = "Person 0"
    elif 69 <= i <= 83:
        person = "Person 2"
    elif (84 <= i <= 114) and (380 <= i <= 400):
        person = "Person 3"
    elif (115 <= i <= 141) and (312 <= i <= 363):
        person = "Person 4"
    elif (142 <= i <= 190):
        person = "Person 5"
    elif (191 <= i <= 200) and (251 <= i <= 311) :
        person = "Person 6"
    elif (201<= i <= 250) and (364 <= i <= 379):
        person = "Person 7"
    else :
        person = "Person 1"
    ground_truth[filename] = person

print(ground_truth)

# Initialize variables for accuracy calculation
total_images = len(ground_truth)
print(total_images)
correct_predictions = 0
X = [] # Face encodings
y = [] # Person IDs

# Load the known image
image_of_person = face_recognition.load_image_file("test_5.jpg")
known_face_encoding =
face_recognition.face_encodings(image_of_person)[0] true=0

# Iterate over each known image
```

```
for filename, person_id in ground_truth.items():
    # Load the unknown image
    unknown_face_image =
    face_recognition.load_image_file(filename) faces =
    face_recognition.face_locations(unknown_face_image) if
    len(faces) == 0:
        print(f"No faces found in image: {filename}")
        continue
    unknown_face_encoding =
face_recognition.face_encodings(unknown_face_image)[0]
    X.append(unknown_face_encoding)
    y.append(person_id)
    # Compare the face encodings
    results =
face_recognition.compare_faces([known_face_encoding],
unknown_face_encoding, tolerance=0.6)
    print(results)
    if (results[0]=="True" and person_id=="Person 1"):
      true+=1
    if (results[0]=="False" and person_id != "Person 1"):
      true +=1
    # Check if the prediction matches the ground truth
    if results[0]:
        correct_predictions += 1
# Calculate accuracy
print(correct_predictions)
accuracy_img = (correct_predictions / total_images) * 100
print(f"Accuracy of Predicting The Test Image From the Dataset:
{accuracy_img:.2f}%")

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.5, random_state=42)

# Initialize and train the classifier (Random Forest as an
example) clf = RandomForestClassifier(n_estimators=100,
random_state=42) clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)
print(y_train)
print(y_pred)
print(y_test)
# Calculate accuracy
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
matrix = classification_report(y_test,y_pred)
print("Classification Report")
print(matrix)
# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)
# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

CNN MODEL FOR CLASSIFICATION:

```
import numpy as np
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import
Conv2D,MaxPooling2D,Dense,Flatten,Dropout import
matplotlib.pyplot as plt
from tensorflow.keras.layers import
BatchNormalization from
tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import
ImageDataGenerator from sklearn.model_selection import
train_test_split import os

from google.colab import drive
drive.mount('/content/drive')
train_dir = "/content/drive/MyDrive/IMAGES"
generator = ImageDataGenerator()
```

```
train_ds =
generator.flow_from_directory(train_dir,target_size=(224,
224),batch_size=32)
classes = list(train_ds.class_indices.keys())
model = Sequential()
model.add(Conv2D(32, kernel_size = (3, 3), activation='relu',
input_shape=(224,224,3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(96, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
#model.add(Dropout(0.3))
model.add(Dense(len(classes),activation='softmax'))

history = model.fit(train_ds,epochs= 10, batch_size=16)

plt.plot(history.history['accuracy'])
plt.plot(history.history['loss'])
plt.xlabel('Time')
plt.legend(['accuracy', 'loss'])
plt.show()

import tensorflow as tf
from tensorflow.keras.preprocessing import image

def predict_image(image_path):
    img =
tf.keras.preprocessing.image.load_img(image_path,
target_size=(224, 224,3))
    plt.imshow(img)
```

```
    plt.axis('off')
    plt.show()
    x = tf.keras.preprocessing.image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    pred = model.predict(x)
    print("Actual: " + (image_path.split("/")[-1]).split("_")[0])
    print("Predicted: " + classes[np.argmax(pred)])

predict_image("/content/drive/MyDrive/MAIN/NANI/test_6.jpg")
```

REFERENCES:

• Low-dimensional procedure for the characterization of human faces by L.Sirovich and M.Kirby • Face Recognition using Eigenfaces by Matthew A. Turk and Alex P.Pentland • Haar Cascade Frontal Face Detection