```
!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")
```