import os

import cv2

import numpy as np

import torch

from facenet\_pytorch import InceptionResnetV1, MTCNN

# Initialize MTCNN and InceptionResnetV1

mtcnn = MTCNN(keep\_all=True)

resnet = InceptionResnetV1(pretrained='vggface2').eval()

# Function to detect and encode faces

def detect\_and\_encode(image):

    with torch.no\_grad():

        boxes, \_ = mtcnn.detect(image)

        if boxes is not None:

            faces = []

            for box in boxes:

                face = image[int(box[1]):int(box[3]), int(box[0]):int(box[2])]

                if face.size == 0:

                    continue

                face = cv2.resize(face, (160, 160))

                face = np.transpose(face, (2, 0, 1)).astype(np.float32) / 255.0

                face\_tensor = torch.tensor(face).unsqueeze(0)

                encoding = resnet(face\_tensor).detach().numpy().flatten()

                faces.append(encoding)

            return faces

    return []

# Function to encode specific images with predefined names

def encode\_known\_faces(known\_faces):

    known\_face\_encodings = []

    known\_face\_names = []

    for name, image\_path in known\_faces.items():

        known\_image = cv2.imread(image\_path)

        if known\_image is not None:

            known\_image\_rgb = cv2.cvtColor(known\_image, cv2.COLOR\_BGR2RGB)

            encodings = detect\_and\_encode(known\_image\_rgb)

            if encodings:

                known\_face\_encodings.append(encodings[0])  # Assuming one face per image

                known\_face\_names.append(name)

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

# Define known faces with explicit names

known\_faces = {

    "Carlos Sainz": "C:/Users/Acer/.vscode/Projects/carlos-sainz-ferrari.jpg",

    "Lando Norris": "C:/Users/Acer/.vscode/Projects/lando\_norris.jpg",

    "Charles Leclerc": "C:/Users/Acer/.vscode/Projects/charles\_leclerc.jpg"

}

# Encode known faces

known\_face\_encodings, known\_face\_names = encode\_known\_faces(known\_faces)

# Function to recognize faces

def recognize\_faces(known\_encodings, known\_names, test\_encodings, threshold=0.6):

    recognized\_names = []

    for test\_encoding in test\_encodings:

        distances = np.linalg.norm(known\_encodings - test\_encoding, axis=1)

        min\_distance\_idx = np.argmin(distances)

        if distances[min\_distance\_idx] < threshold:

            recognized\_names.append(known\_names[min\_distance\_idx])

        else:

            recognized\_names.append('Not Recognized')

    return recognized\_names

# Start video capture

cap = cv2.VideoCapture(0)

threshold = 0.6

while cap.isOpened():

    ret, frame = cap.read()

    if not ret:

        break

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

    test\_face\_encodings = detect\_and\_encode(frame\_rgb)

    if test\_face\_encodings and known\_face\_encodings:

        names = recognize\_faces(np.array(known\_face\_encodings), known\_face\_names, test\_face\_encodings, threshold)

        for name, box in zip(names, mtcnn.detect(frame\_rgb)[0]):

            if box is not None:

                (x1, y1, x2, y2) = map(int, box)

                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

                cv2.putText(frame, name, (x1, y1 - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE\_AA)

    cv2.imshow('Face Recognition', frame)

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

        break

cap.release()

cv2.destroyAllWindows()

This code is a **face recognition application** that uses a webcam to detect and recognize faces in real-time. Below is a step-by-step explanation:

**1. Import Libraries**

import os

import cv2

import numpy as np

import torch

from facenet\_pytorch import InceptionResnetV1, MTCNN

* **os**: Used for file and directory operations.
* **cv2**: OpenCV for video capture and image processing.
* **numpy**: Handles arrays for numerical operations.
* **torch**: PyTorch library for deep learning models.
* **facenet\_pytorch**: Provides MTCNN (face detector) and InceptionResnetV1 (face recognition model).

**2. Initialize Models**

mtcnn = MTCNN(keep\_all=True)

resnet = InceptionResnetV1(pretrained='vggface2').eval()

* **MTCNN**: A multi-task convolutional neural network used for face detection.
* **InceptionResnetV1**: A pre-trained face recognition model based on the VGGFace2 dataset, used to generate face encodings.

**3. Detect and Encode Faces**

def detect\_and\_encode(image):

with torch.no\_grad():

boxes, \_ = mtcnn.detect(image) # Detect faces

if boxes is not None:

faces = []

for box in boxes:

face = image[int(box[1]):int(box[3]), int(box[0]):int(box[2])] # Extract face region

if face.size == 0:

continue

face = cv2.resize(face, (160, 160)) # Resize to model's input size

face = np.transpose(face, (2, 0, 1)).astype(np.float32) / 255.0 # Normalize and transpose

face\_tensor = torch.tensor(face).unsqueeze(0) # Convert to tensor

encoding = resnet(face\_tensor).detach().numpy().flatten() # Generate encoding

faces.append(encoding)

return faces

return []

* Detects faces in an image, extracts them, resizes, normalizes, and converts them into embeddings (numerical representations).

**4. Encode Known Faces**

def encode\_known\_faces(known\_faces):

known\_face\_encodings = []

known\_face\_names = []

for name, image\_path in known\_faces.items():

known\_image = cv2.imread(image\_path) # Load image

if known\_image is not None:

known\_image\_rgb = cv2.cvtColor(known\_image, cv2.COLOR\_BGR2RGB) # Convert to RGB

encodings = detect\_and\_encode(known\_image\_rgb) # Get face encoding

if encodings:

known\_face\_encodings.append(encodings[0]) # Assume one face per image

known\_face\_names.append(name)

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

* Reads images of known individuals, extracts their face encodings, and stores them with their names.

**5. Known Faces**

known\_faces = {

"Carlos Sainz": "C:/Users/Acer/.vscode/Projects/carlos-sainz-ferrari.jpg",

"Lando Norris": "C:/Users/Acer/.vscode/Projects/lando\_norris.jpg",

"Charles Leclerc": "C:/Users/Acer/.vscode/Projects/charles\_leclerc.jpg"

}

known\_face\_encodings, known\_face\_names = encode\_known\_faces(known\_faces)

* Specifies known individuals with their image paths. Encodes these faces for later recognition.

**6. Recognize Faces**

def recognize\_faces(known\_encodings, known\_names, test\_encodings, threshold=0.6):

recognized\_names = []

for test\_encoding in test\_encodings:

distances = np.linalg.norm(known\_encodings - test\_encoding, axis=1) # Calculate distance

min\_distance\_idx = np.argmin(distances) # Find closest match

if distances[min\_distance\_idx] < threshold: # Threshold for recognition

recognized\_names.append(known\_names[min\_distance\_idx])

else:

recognized\_names.append('Not Recognized')

return recognized\_names

* Compares test face encodings with known encodings using Euclidean distance. Recognizes if the distance is below a specified threshold.

**7. Real-Time Face Recognition with Webcam**

**a. Start Video Capture**

cap = cv2.VideoCapture(0)

threshold = 0.6

* Opens the webcam (device 0) for real-time video capture.
* Sets a threshold for face recognition accuracy.

**b. Process Each Frame**

while cap.isOpened():

ret, frame = cap.read() # Read video frame

if not ret:

break

frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB) # Convert to RGB

test\_face\_encodings = detect\_and\_encode(frame\_rgb) # Detect and encode faces

* Reads frames from the webcam and processes them for face detection and encoding.

**c. Recognize Faces in the Frame**

if test\_face\_encodings and known\_face\_encodings:

names = recognize\_faces(np.array(known\_face\_encodings), known\_face\_names, test\_face\_encodings, threshold)

for name, box in zip(names, mtcnn.detect(frame\_rgb)[0]):

if box is not None:

(x1, y1, x2, y2) = map(int, box)

cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2) # Draw rectangle

cv2.putText(frame, name, (x1, y1 - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE\_AA)

* Recognizes faces in the current frame by comparing test encodings with known encodings.
* Draws a rectangle around recognized faces and displays the name.

**d. Display Video Feed**

cv2.imshow('Face Recognition', frame)

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

break

* Displays the video feed with recognized faces. Pressing q exits the loop.

**e. Release Resources**

cap.release()

cv2.destroyAllWindows()

* Stops video capture and closes all OpenCV windows.

**Summary**

* **Detect and encode faces** using MTCNN and InceptionResnetV1.
* **Compare encodings** to recognize faces based on a threshold.
* Displays a **real-time video feed** with bounding boxes and labels for recognized faces.