SESSION 4

TOC:

1. Face Recognition Library:

1.1 Overview of popular face recognition libraries:

OpenCV:
OpenCV is a widely-used computer vision library that provides various functionalities, including face detection and recognition. It has robust support for image and video processing, making it a popular choice for face recognition tasks.
dlib:
dlib is a C++ library with Python bindings that offers advanced face detection and recognition capabilities. It provides pre-trained models for face landmark detection and facial feature extraction, making it useful for various face analysis tasks.
face_recognition:
face_recognition is a high-level face recognition library built on top of dlib. It simplifies the face recognition process by providing a simple API for face detection, face encoding, and face comparison.

Installation and setup of the chosen face recognition library:

To install a specific face recognition library, you can use the following commands:

OPENCV :

In [ ]:
```
!pip install opencv-python
```

```
Requirement already satisfied: opencv-python in c:\users\omolp091\anaconda3\lib\site-packages (4.8.1.78)
Requirement already satisfied: numpy>=1.17.3 in c:\users\omolp091\anaconda3\lib\site-packages (from opencv-python) (1.23.5)
```

dlib:

In [ ]:
```
!pip install dlib
```

```
Requirement already satisfied: dlib in c:\users\omolp091\anaconda3\lib\site-packages (19.24.2)
```

face_recognition:

In [ ]:
```
!pip install face-recognition
```

```
Requirement already satisfied: face-recognition in c:\users\omolp091\anaconda3\lib\site-packages (1.3.0)
Requirement already satisfied: dlib>=19.7 in c:\users\omolp091\anaconda3\lib\site-packages (from face-recognition) (19.24.2)
Requirement already satisfied: Pillow in c:\users\omolp091\anaconda3\lib\site-packages (from face-recognition) (9.4.0)
Requirement already satisfied: numpy in c:\users\omolp091\anaconda3\lib\site-packages (from face-recognition) (1.23.5)
Requirement already satisfied: face-recognition-models>=0.3.0 in c:\users\omolp091\anaconda3\lib\site-packages (from face-recogn
ition) (0.3.0)
Requirement already satisfied: Click>=6.0 in c:\users\omolp091\anaconda3\lib\site-packages (from face-recognition) (8.0.4)
Requirement already satisfied: colorama in c:\users\omolp091\anaconda3\lib\site-packages (from Click>=6.0->face-recognition) (0.
4.6)
```

Introduction to the library's features and capabilities: Each face recognition library offers different features and capabilities. Here's a brief overview of what you can expect from each library:

OpenCV: OpenCV provides functions for face detection using Haar cascades or deep learning models. It also offers utilities for face recognition, such as Eigenfaces and Fisherfaces, and the ability to train custom models.

dlib: dlib offers state-of-the-art face detection and recognition algorithms. It provides pre-trained models for face landmark detection, facial feature extraction, and face recognition. It also supports face clustering and face alignment.

face_recognition: face_recognition library simplifies face recognition tasks by providing a high-level API. It utilizes dlib's face recognition models for face detection, encoding, and comparison. It allows you to compare faces, identify faces in images, and perform facial feature extraction.

face_recognition: face_recognition library simplifies face recognition tasks by providing a high-level API. It utilizes dlib's face recognition models for face detection, encoding, and comparison. It allows you to compare faces, identify faces in images, and perform facial feature extraction.face_recognition: face_recognition library simplifies face recognition tasks by providing a high-level API. It utilizes dlib's face recognition models for face detection, encoding, and comparison. It allows you to compare faces, identify faces in images, and perform facial feature extraction.

1.2 Introduction to face detection algorithms and techniques:

Face detection is a fundamental task in computer vision that involves locating and identifying faces within an image or video. Several algorithms and techniques have been developed for this purpose. Here are some commonly used ones:

Haar cascades: Haar cascades are a machine learning-based face detection algorithm. They use a cascade of simple classifiers trained on Haar-like features to detect faces. Haar-like features are rectangular regions with specific intensity patterns, such as edges and gradients. Haar cascades are

fast and efficient, making them suitable for real-time face detection.

Deep learning-based detectors: Deep learning approaches have achieved significant advancements in face detection. These methods employ convolutional neural networks (CNNs) to learn discriminative features and make accurate predictions. Popular deep learning-based face detectors include the Single Shot Multibox Detector (SSD) and the You Only Look Once (YOLO) algorithm. These detectors offer higher accuracy but may be computationally more expensive.

Understanding the Haar cascades algorithm and its implementation using OpenCV: Haar cascades algorithm can be implemented using the OpenCV library. Here's an example of using Haar cascades for face detection:

In [ ]:
```
import cv2
```

Load the pre-trained Haar cascade XML file

In [ ]:
```
face_cascade = cv2.CascadeClassifier('images/haarcascade_frontalface_default.xml')
```

Load the input image

In [ ]:
```
image = cv2.imread('Images/input_image.jpg')
```

Convert the image to grayscale

In [ ]:
```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Perform face detection

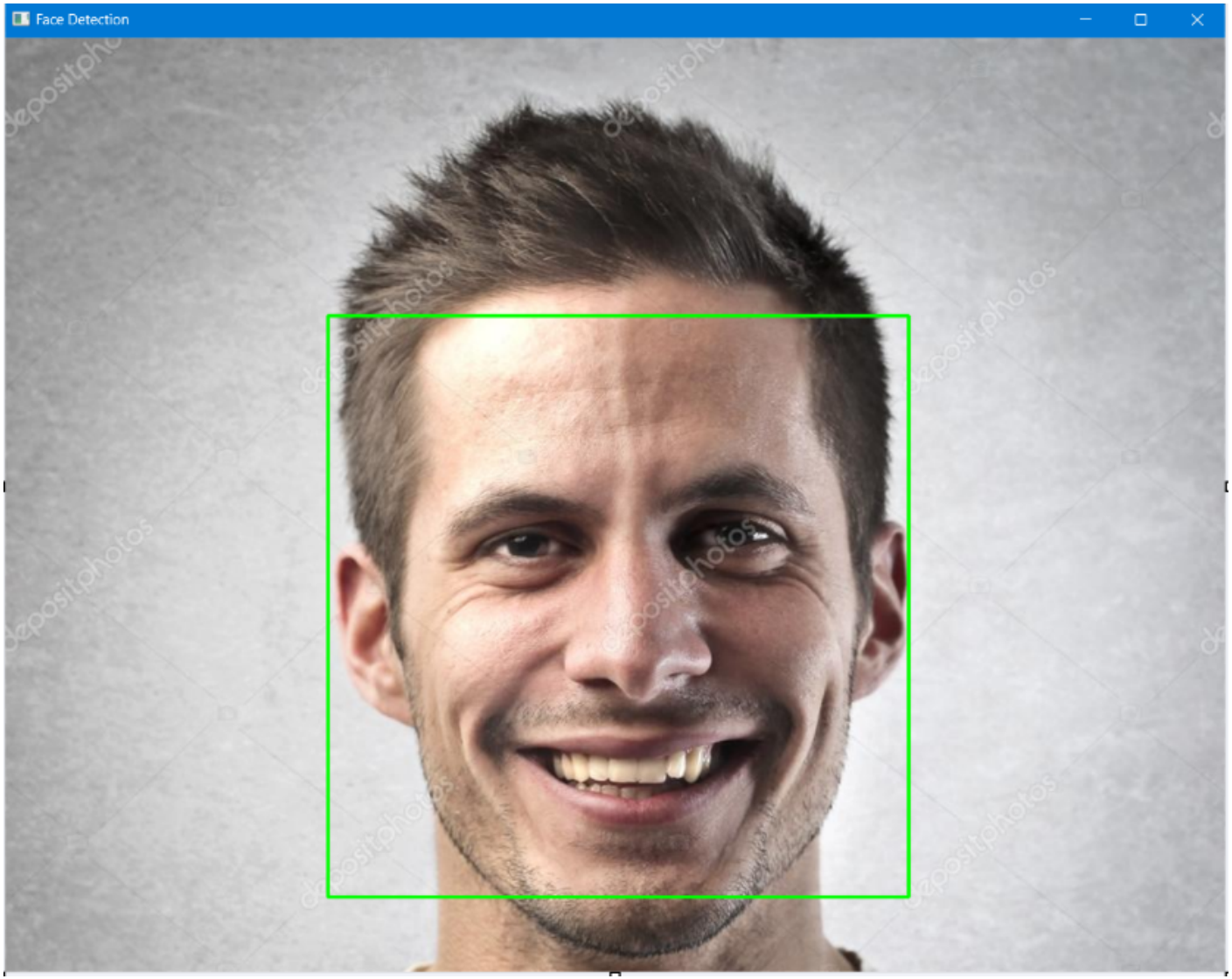
In [ ]:
```
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
```

Draw bounding boxes around the detected faces

In [ ]:
```
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)
```

Display the result

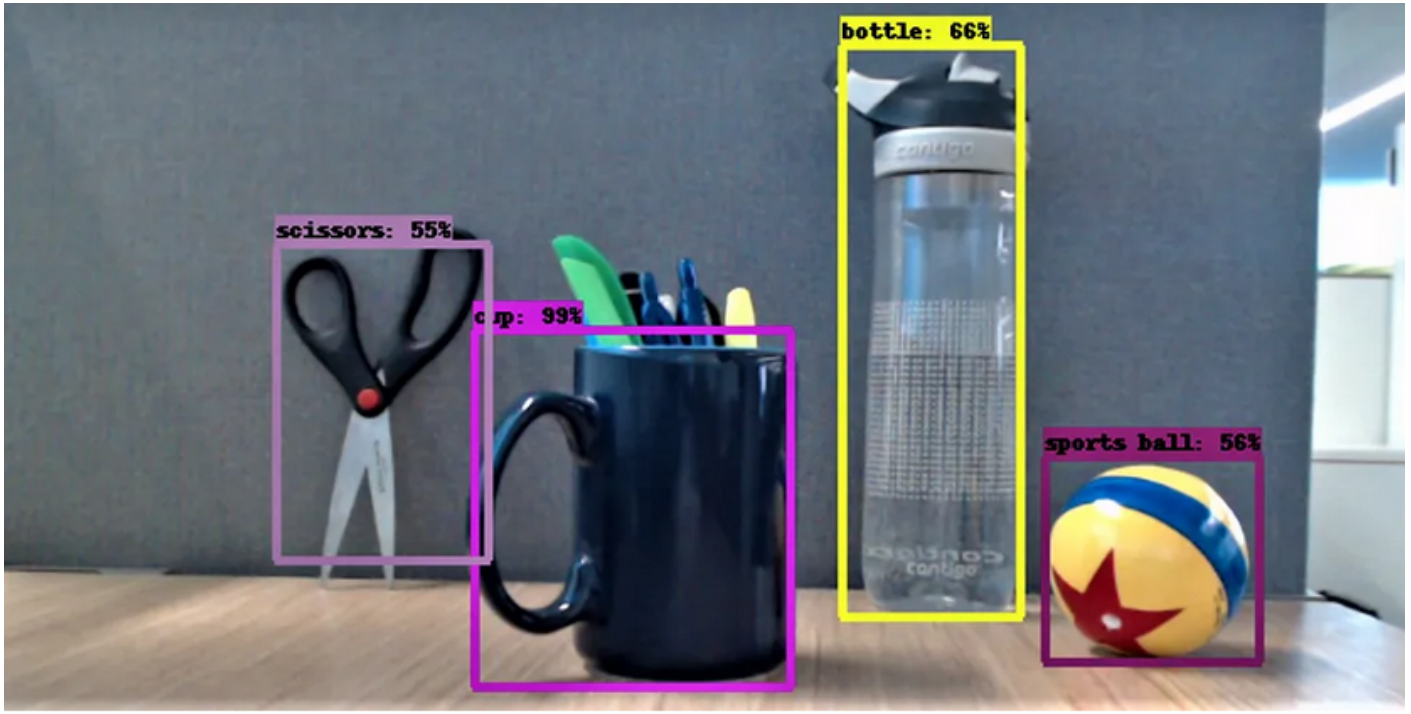
In [ ]:
```
cv2.imshow('Face Detection', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



SSD Object Detection in Real Time (Deep Learning and Caffe)

Utilizing deep learning-based face detectors like the SSD or YOLO algorithm: Deep learning-based face detectors, such as SSD and YOLO, provide higher accuracy compared to Haar cascades but require more computational resources. Here's an example of using the SSD face detector with OpenCV's DNN module:

In this section, we will be seeing SSD Object Detection- features, advantages, drawbacks, and implement MobileNet SSD model with Caffe — using OpenCV in Python.



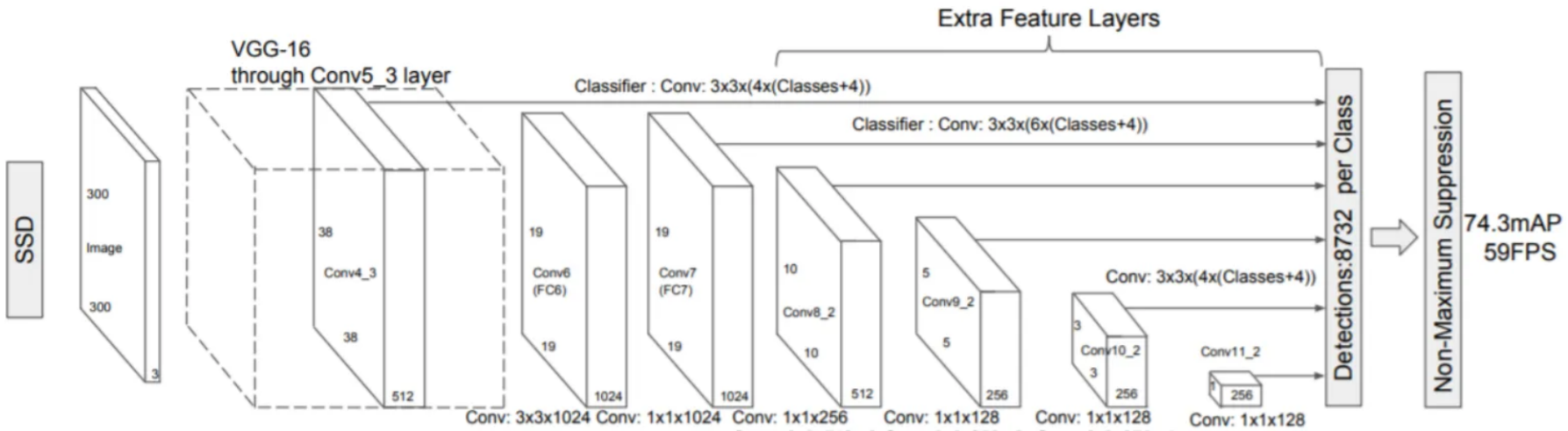
Real Time Object Detection

What is Object Detection?

Object Detection in Computer Vision is as simple as it sounds- detecting and predicting objects and localizing their area. Object Detection is based on image classification. Irrespective of the latter being performed using neural networks or primitive classifiers, image classification is always the first step. Building further on this, we can perform detection which localizes all possible objects in a given frame.

Single Shot MultiBox Detector (SSD)

SSD Object Detection extracts feature map using a base deep learning network, which are CNN based classifiers, and applies convolution filters to finally detect objects. Our implementation uses MobileNet as the base network (others might include- VGGNet, ResNet, DenseNet).



efficient alternative to other frameworks to perform object detection. Caffe

MobileNet SSD object detection using OpenCV 3.4.1 DNN module

Let us implement how to use the OpenCV 3.4.1 deep learning module with the MobileNet-SSD network for object discovery.

As part of Opencv 3.4. + The deep neural network (DNN) module was officially included. The DNN module allows loading pre-trained models of most popular deep learning frameworks, including Tensorflow, Caffe, Darknet, Torch. Besides MobileNet-SDD, other architectures are compatible with OpenCV 3.4.1:

```
GoogleLeNet
YOLO
SqueezeNet
R-CNN faster
ResNet
This API is compatible with C ++ and Python
```

What is Caffe?

Caffe is a deep learning framework developed by Berkeley AI Research and community contributors. Caffe was developed as a faster and far more efficient alternative to other frameworks to perform object detection. Caffe can process 60 million images per day with a single NVIDIA K-40 GPU. That is 1 ms/image for inference and 4 ms/image for learning.

```
Descripton code
```

In this section, we will create the Python script for object detection and explain, how to load our deep neural network with OpenCV 3.4? How to pass the image to the neural network? and How to make a prediction with MobileNet or dnn module in OpenCV ?.

We use a pre-trained MobileNet taken from https://github.com/chuanqi305/MobileNet-SSD/ that was trained on the Caffe-SSD framework. This model can detect 2 classes.

Load and predict with the deep neural network module

Import the Libraries

In [ ]:
```
import numpy as np
import argparse
import cv2
```

construct the argument parse

In [ ]:
```
# parser = argparse.ArgumentParser(
#     description='Script to run MobileNet-SSD object detection network ')
# parser.add_argument("--video", help="path to video file. If empty, camera's stream will be used")
# parser.add_argument("--prototxt", default="MobileNetSSD_deploy.prototxt",
#                                   help='Path to text network file: '
#                                        'MobileNetSSD_deploy.prototxt for Caffe model or '
#                                       )
# parser.add_argument("--weights", default="MobileNetSSD_deploy.caffemodel",
#                                  help='Path to weights: '
#                                       'MobileNetSSD_deploy.caffemodel for Caffe model or '
#                                      )
```

```
# parser.add_argument("--thr", default=0.2, type=float, help="confidence threshold to filter out weak detections")
# args = parser.parse_args()
```

Load the Caffe model

In [ ]:
```
# net = cv2.dnn.readNetFromCaffe(args.prototxt, args.weights)
```

Capture frame-by-frame

In [ ]:
```
# ret, frame = cap.read()
# frame_resized = cv2.resize(frame,(300,300)) # resize frame for prediction
```

MobileNet requires fixed dimensions for input image(s)

so we have to ensure that it is resized to 300x300 pixels.
set a scale factor to image because network the objects has differents size.
We perform a mean subtraction (127.5, 127.5, 127.5) to normalize the input;
after executing this command our "blob" now has the shape:
(1, 3, 300, 300)

In [ ]:
```
    # MobileNet requires fixed dimensions for input image(s)
    # so we have to ensure that it is resized to 300x300 pixels.
    # set a scale factor to image because network the objects has differents size.
    # We perform a mean subtraction (127.5, 127.5, 127.5) to normalize the input;
    # after executing this command our "blob" now has the shape:
    # (1, 3, 300, 300)
       #  blob = cv2.dnn.blobFromImage(frame_resized, 0.007843, (300, 300), (127.5, 127.5, 127.5), False)
    #Set to network the input blob
       #  net.setInput(blob)
    #Prediction of network
       #  detections = net.forward()
```

Complete Code

In [ ]:
```
import numpy as np
import cv2

# Set the values directly in the notebook
```

```
args = {
    "video": "",  # Set the path to the video file or leave it empty for camera stream
    "prototxt": "Images/MobileNetSSD_deploy.prototxt",
    "weights": "Images/MobileNetSSD_deploy.caffemodel",
    "thr": 0.2
}

# Labels of Network.
classNames = {0: 'background', 1: 'aeroplane', 2: 'bicycle', 3: 'bird', 4: 'boat',
              5: 'bottle', 6: 'bus', 7: 'car', 8: 'cat', 9: 'chair',
              10: 'cow', 11: 'diningtable', 12: 'dog', 13: 'horse',
              14: 'motorbike', 15: 'person', 16: 'pottedplant',
              17: 'sheep', 18: 'sofa', 19: 'train', 20: 'tvmonitor'}

# Open video file or capture device.
if args["video"]:
    cap = cv2.VideoCapture(args["video"])
else:
    cap = cv2.VideoCapture(0)

# Load the Caffe model
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["weights"])

while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
    frame_resized = cv2.resize(frame, (300, 300))  # resize frame for prediction

    # MobileNet requires fixed dimensions for input image(s)
    blob = cv2.dnn.blobFromImage(frame_resized, 0.007843, (300, 300), (127.5, 127.5, 127.5), False)
    net.setInput(blob)
    detections = net.forward()

    cols = frame_resized.shape[1]
    rows = frame_resized.shape[0]

    for i in range(detections.shape[2]):
        confidence = detections[0, 0, i, 2]
        if confidence > args["thr"]:
            class_id = int(detections[0, 0, i, 1])

            xLeftBottom = int(detections[0, 0, i, 3] * cols)
            yLeftBottom = int(detections[0, 0, i, 4] * rows)
            xRightTop = int(detections[0, 0, i, 5] * cols)
```

```
            yRightTop = int(detections[0, 0, i, 6] * rows)

            heightFactor = frame.shape[0] / 300.0
            widthFactor = frame.shape[1] / 300.0
            xLeftBottom = int(widthFactor * xLeftBottom)
            yLeftBottom = int(heightFactor * yLeftBottom)
            xRightTop = int(widthFactor * xRightTop)
            yRightTop = int(heightFactor * yRightTop)

            cv2.rectangle(frame, (xLeftBottom, yLeftBottom), (xRightTop, yRightTop), (0, 255, 0))

            if class_id in classNames:
                label = classNames[class_id] + ": " + str(confidence)
                labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.5, 1)

                yLeftBottom = max(yLeftBottom, labelSize[1])
                cv2.rectangle(frame, (xLeftBottom, yLeftBottom - labelSize[1]),
                              (xLeftBottom + labelSize[0], yLeftBottom + baseLine),
                              (255, 255, 255), cv2.FILLED)
                cv2.putText(frame, label, (xLeftBottom, yLeftBottom),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))

                print(label)

    cv2.namedWindow("frame", cv2.WINDOW_NORMAL)
    cv2.imshow("frame", frame)
    if cv2.waitKey(1) >= 0:  # Break with ESC
        break

# Release the video capture object and destroy all windows
cap.release()
cv2.destroyAllWindows()
```

```
person: 0.99999845
person: 0.9658812
person: 0.99999726
person: 0.9248693
chair: 0.31879643
person: 0.99999666
person: 0.91995615
chair: 0.33836967
person: 0.9999964
person: 0.91965336
chair: 0.40895724
person: 0.9999968
person: 0.94828403
chair: 0.3773629
person: 0.9999957
person: 0.9485657
person: 0.99999654
person: 0.9647959
person: 0.99999714
person: 0.96207786
person: 0.9999974
person: 0.98159516
person: 0.9999969
person: 0.98072493
person: 0.9999981
person: 0.97760016
chair: 0.2569076
person: 0.9999981
person: 0.98178333
person: 0.99999857
person: 0.97876215
person: 0.9999982
person: 0.9790496
person: 0.99999857
person: 0.98625827
chair: 0.28663898
person: 0.99999833
person: 0.9860406
chair: 0.34131056
person: 0.99999845
person: 0.99070823
chair: 0.26636866
person: 0.9999981
person: 0.98625064
```

```
chair: 0.28102937
person: 0.9999981
person: 0.9742708
chair: 0.3555592
person: 0.9999982
person: 0.97869337
chair: 0.2959008
person: 0.9999982
person: 0.9795997
chair: 0.35975054
person: 0.99999845
person: 0.9785993
chair: 0.37686333
person: 0.9999981
person: 0.9564113
chair: 0.37502858
person: 0.9999982
person: 0.9504963
chair: 0.30369422
person: 0.999998
person: 0.9508912
chair: 0.3677565
person: 0.999998
person: 0.9606987
chair: 0.32737345
person: 0.9999981
person: 0.96580005
chair: 0.37768632
person: 0.99999785
person: 0.9639838
chair: 0.35919055
person: 0.999998
person: 0.97024363
chair: 0.30242568
person: 0.99999785
person: 0.9762241
chair: 0.35003743
person: 0.99999785
person: 0.9842418
chair: 0.2929935
person: 0.99999774
person: 0.98365605
chair: 0.36459088
person: 0.2564383
```

```
person: 0.99999726
person: 0.9877251
chair: 0.33004135
person: 0.29024506
person: 0.99999714
person: 0.98725116
chair: 0.41859287
person: 0.27055094
person: 0.99999726
person: 0.9896797
chair: 0.46551087
person: 0.3085961
person: 0.9999964
person: 0.9927632
person: 0.34953803
chair: 0.3232757
person: 0.99999595
person: 0.99240017
person: 0.34489426
chair: 0.33905813
person: 0.99999607
person: 0.99301434
chair: 0.43612581
person: 0.41904625
person: 0.99999595
person: 0.9909733
person: 0.44275746
chair: 0.36346024
person: 0.99999607
person: 0.9904174
person: 0.4182858
chair: 0.38234773
person: 0.9999943
person: 0.959234
person: 0.42566812
chair: 0.3167486
person: 0.9999943
person: 0.96093136
person: 0.32814085
person: 0.99999166
person: 0.9645138
chair: 0.3369099
person: 0.321087
person: 0.9999907
```

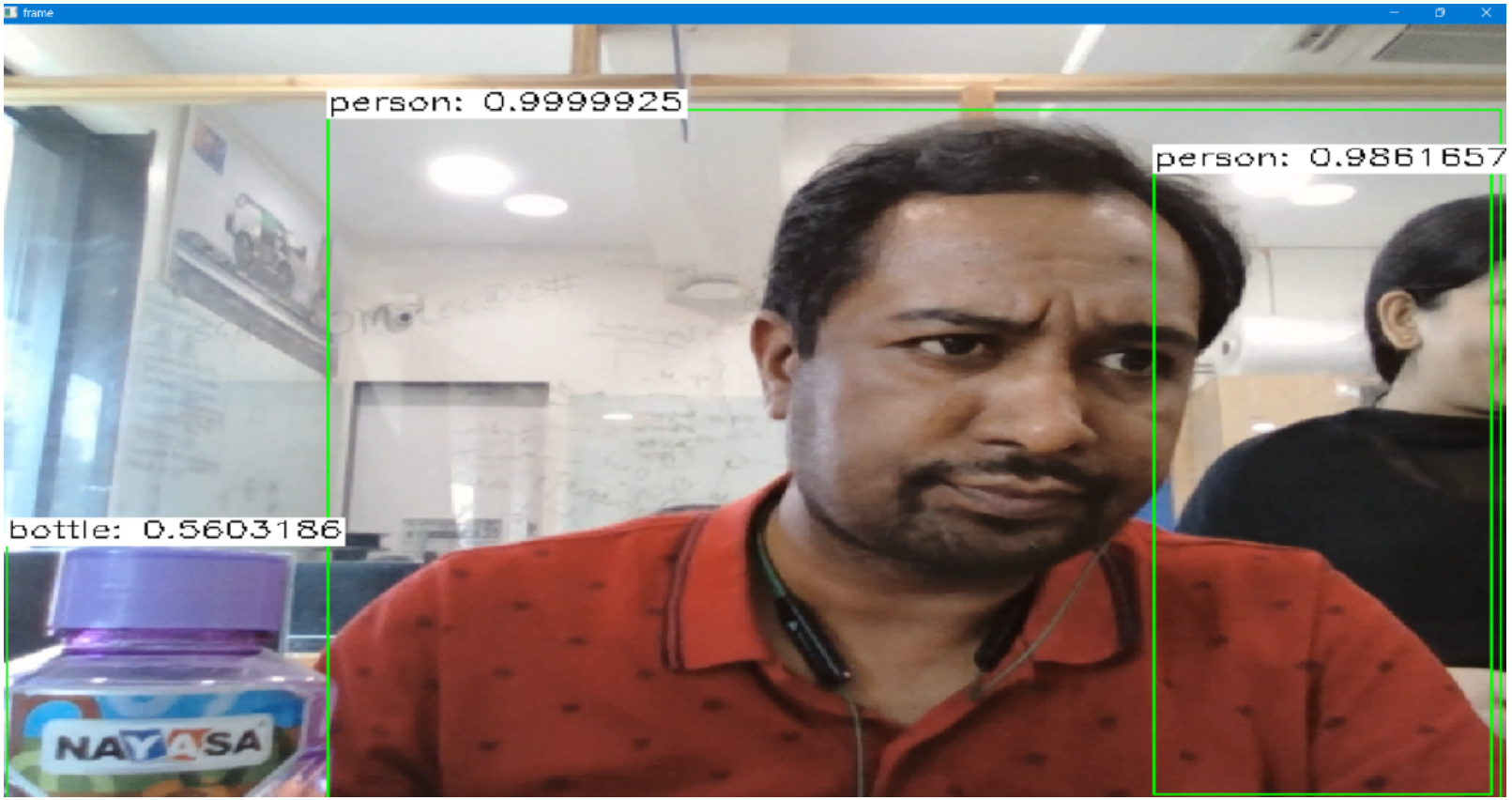
```
person: 0.96414995
chair: 0.35721588
person: 0.33952916
person: 0.9999864
person: 0.9723112
person: 0.31293678
chair: 0.26969638
person: 0.99998415
person: 0.97757506
chair: 0.3168085
person: 0.3157822
person: 0.999977
person: 0.9854218
person: 0.36462188
chair: 0.3018313
person: 0.9999832
person: 0.9856023
person: 0.34240338
chair: 0.28004482
diningtable: 0.2779526
person: 0.99998426
person: 0.9853807
person: 0.4243424
diningtable: 0.28181157
chair: 0.25282633
person: 0.9999862
person: 0.98232377
person: 0.38647264
chair: 0.27127606
diningtable: 0.2672748
person: 0.9999864
person: 0.9835361
person: 0.34000483
diningtable: 0.26863265
person: 0.99998975
person: 0.97551763
person: 0.38678455
chair: 0.3150682
person: 0.9999894
person: 0.9747902
person: 0.43699384
chair: 0.34607282
diningtable: 0.2601914
person: 0.9999882
```

```
person: 0.9638729
person: 0.46269223
diningtable: 0.30551448
chair: 0.29120576
person: 0.9999852
person: 0.97126555
person: 0.42736387
chair: 0.39694953
diningtable: 0.3588205
person: 0.9999839
person: 0.96316683
person: 0.3801932
chair: 0.36219522
diningtable: 0.32837048
person: 0.99998426
person: 0.94416153
chair: 0.48800436
person: 0.3904268
diningtable: 0.26316404
person: 0.99998045
person: 0.93906295
person: 0.5199018
chair: 0.43543112
diningtable: 0.3015363
person: 0.9999827
person: 0.9441642
chair: 0.4300096
person: 0.42085302
diningtable: 0.31839657
person: 0.9999695
person: 0.94723284
chair: 0.43759102
diningtable: 0.37249637
person: 0.3535692
person: 0.9999751
person: 0.9392186
chair: 0.46579397
diningtable: 0.34882733
person: 0.339058
person: 0.9999808
person: 0.93658084
chair: 0.53437877
person: 0.35273832
diningtable: 0.33996427
```

```
person: 0.9999826
person: 0.9492988
chair: 0.4255999
diningtable: 0.35232148
person: 0.2917588
person: 0.9999809
person: 0.94727594
chair: 0.38917848
diningtable: 0.35939455
person: 0.31212157
person: 0.99998367
person: 0.9461379
chair: 0.4192173
diningtable: 0.3677801
person: 0.99998283
person: 0.944883
diningtable: 0.31494895
chair: 0.30882594
person: 0.26315424
person: 0.99999034
person: 0.9467055
chair: 0.38069308
diningtable: 0.2982285
diningtable: 0.2561001
person: 0.99999213
person: 0.9506842
chair: 0.44660684
person: 0.99999166
person: 0.94968003
chair: 0.41375157
person: 0.9999926
person: 0.95658666
chair: 0.4296086
person: 0.9999939
person: 0.9471407
chair: 0.40482348
diningtable: 0.2767555
person: 0.9999939
person: 0.9508483
chair: 0.31972232
diningtable: 0.28260568
person: 0.9999957
person: 0.9590769
chair: 0.33357716
```

```
diningtable: 0.27549744
person: 0.9999958
person: 0.96654373
chair: 0.45138735
person: 0.32259068
diningtable: 0.2751309
person: 0.99999464
person: 0.9792901
person: 0.53646135
chair: 0.42371547
diningtable: 0.26913017
person: 0.99999464
person: 0.98036
person: 0.52692634
chair: 0.42586288
diningtable: 0.28074923
person: 0.99999607
person: 0.97334665
chair: 0.43580702
person: 0.4209273
person: 0.9999968
person: 0.9747422
chair: 0.44320637
person: 0.4192263
diningtable: 0.25579038
person: 0.999997
person: 0.9713587
chair: 0.46969494
person: 0.37005025
diningtable: 0.28056997
person: 0.999997
person: 0.975764
person: 0.43423647
chair: 0.41290388
diningtable: 0.28673548
person: 0.9999962
person: 0.9686321
person: 0.42248818
chair: 0.41409254
diningtable: 0.2832665
person: 0.999997
person: 0.9729567
person: 0.47921023
chair: 0.4158182
```

```
diningtable: 0.29764414
person: 0.9999976
person: 0.9690136
person: 0.43427625
chair: 0.38025913
person: 0.9999975
person: 0.9727638
person: 0.30233133
chair: 0.2608078
```



1.3 Face Encoding:

Overview of face encoding and feature extraction methods: Face encoding, also known as face embedding, is the process of representing a face as a numerical feature vector. This feature vector captures the unique characteristics of a face and can be used for various face recognition tasks.

Here's an overview of face encoding methods:

Geometric-based methods: These methods analyze the geometric properties and spatial relationships between facial landmarks or key points. Features such as distances, angles, or ratios are extracted based on these properties. Statistical models like Principal Component Analysis (PCA) or Linear Discriminant Analysis (LDA) can be applied to further reduce the dimensionality of the features.

Deep learning-based methods: Deep learning approaches have demonstrated remarkable success in face encoding. These methods utilize deep neural networks to directly learn discriminative features from raw face images. By leveraging the power of convolutional neural networks (CNNs), deep learning-based face encoders can capture complex patterns and variations in face appearance.

Introduction to facial landmarks detection and its importance in face encoding: Facial landmarks detection involves identifying key points on a face, such as the corners of the eyes, nose, and mouth. It is important in face encoding because:

Alignment: Facial landmarks can be used to align faces to a standardized pose. By aligning faces, variations due to head pose or facial expression can be minimized, resulting in more accurate feature extraction.

Region of Interest (ROI) selection: Facial landmarks provide information about the structure and shape of a face. They can be used to define a specific region of interest, such as the eyes or mouth, for extracting relevant features.

Using facial landmarks to align faces for accurate feature extraction: To align faces using facial landmarks, you can follow these steps:

Detect facial landmarks using a facial landmarks detection algorithm. Popular libraries like dlib and OpenCV provide pre-trained models for this purpose.

Select specific facial landmarks that define the alignment transformation. For example, you can use the corners of the eyes or the tip of the nose.

Calculate the transformation parameters, such as rotation, translation, and scaling, based on the selected landmarks.

Apply the calculated transformation to align the face to a standardized pose.

Introduction to deep learning-based face encoders like FaceNet, ArcFace, or dlib: Deep learning-based face encoders are powerful methods for extracting face embeddings. Here are brief introductions to some popular face encoding algorithms:

FaceNet: FaceNet is a deep learning-based face recognition model that learns a 128-dimensional embedding for each face. It uses a triplet loss function during training to optimize the embedding space, ensuring that the embeddings of the same person are close together while those of

different people are far apart.

ArcFace: ArcFace is a state-of-the-art deep learning-based face recognition model that introduces an angular margin to the traditional softmax loss. This margin-based loss function enhances the discriminative power of the learned embeddings.

dlib: The dlib library provides a pre-trained face recognition model that combines deep learning with geometric-based alignment. It computes a 128-dimensional face embedding using a neural network trained on a large dataset.

Extracting facial embeddings or feature vectors using the chosen face encoding algorithm: To extract facial embeddings using a chosen face encoding algorithm, you can follow these steps:

Detect and align the face using facial landmarks detection techniques. Preprocess the aligned face image, such as resizing and normalization, to match the requirements of the face encoding model.

Pass the preprocessed face image through the face encoding model to obtain the face embedding. The output will be a high-dimensional feature vector that represents the unique characteristics of the face.

Store the extracted face embeddings in a database or use them for face recognition tasks, such as face identification or verification.

`Here's an example of extracting facial embeddings using the dlib library:`

In [ ]:
```
import dlib
import cv2

# Load the image
image = cv2.imread("Images/input_image.jpg")
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Convert to RGB for dlib

# Load the face landmarks predictor
predictor_path = "Images/shape_predictor_68_face_landmarks.dat"  # Replace with the actual path
predictor = dlib.shape_predictor(predictor_path)

# Load the face detector
detector = dlib.get_frontal_face_detector()

# Detect faces in the image
faces = detector(image_rgb)
```

```
# Iterate over detected faces
for face in faces:
    # Predict face landmarks
    landmarks = predictor(image_rgb, face)
    landmarks_list = [(p.x, p.y) for p in landmarks.parts()]

    # Draw landmarks on the image
    for landmark in landmarks_list:
        cv2.circle(image, landmark, 1, (0, 255, 0), -1)

# Display the image with landmarks
cv2.imshow("Image with Face Landmarks", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

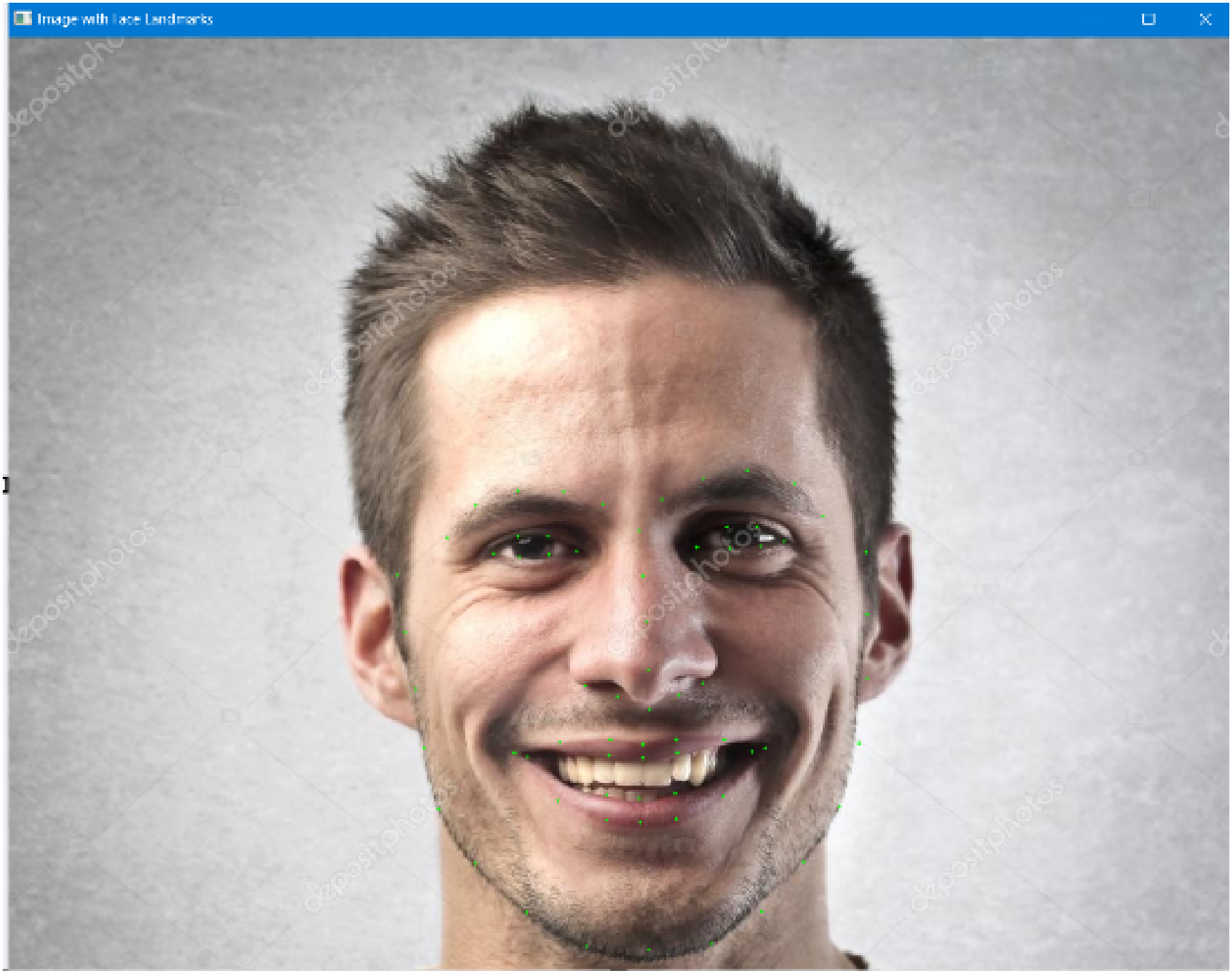


Image with Face Landmarks

1.4 Face Recognition:

Understanding the concept of face recognition and its applications: Face recognition is a technology that identifies or verifies individuals by analyzing and comparing their facial features. It has various applications, including:

Access control: Face recognition can be used for secure authentication and access control in systems such as door entry systems or mobile devices.

Surveillance: Face recognition can aid in identifying individuals in surveillance videos or images, assisting law enforcement agencies in criminal investigations.

Personalization: Face recognition can enable personalized experiences in applications like social media, e-commerce, or personalized advertising.

Creating a face recognition pipeline using the selected face recognition library: To create a face recognition pipeline using a chosen face recognition library, you can follow these steps:

Gather a labeled dataset: Collect a dataset of face images with corresponding labels or identities.

Preprocess the images: Perform necessary preprocessing steps, such as face detection, alignment, resizing, and normalization, to ensure consistent input for the face recognition model.

Train a face recognition model: Utilize the labeled dataset to train a face recognition model. This involves extracting facial features or embeddings and training a classifier or distance-based algorithm for identification or verification.

Implement face identification: Create a function that takes an input face image and compares it with the known faces in the database to identify the person.

Implement face verification: Develop a function that takes an input face image and a claimed identity and determines if the face matches the claimed identity.

Training a face recognition model with a labeled dataset: Here's an example of training a face recognition model using the face_recognition library in Python:

Explanation of the Face Recognition Model in individual steps

Import Libraries

In [ ]:
```
import os
import face_recognition
import numpy as np
import joblib
from PIL import UnidentifiedImageError
```

Path to the labeled dataset

In [ ]:
```
dataset_path = "Images/"
```

Load the dataset

In [ ]:
```
# Load images and compute face encodings
image_paths = [os.path.join(dataset_path, f) for f in os.listdir(dataset_path) if f.lower().endswith(('.png', '.jpg', '.jpeg', '.
known_encodings = []
labels = []
```

Extract face encodings and labels from the dataset

In [ ]:
```
for image_path in image_paths:
    try:
        image = face_recognition.load_image_file(image_path)
        face_encodings = face_recognition.face_encodings(image)

        # Check if at least one face is found
        if face_encodings:
            encoding = face_encodings[0]  # Assuming a single face in each image
            label = os.path.splitext(os.path.basename(image_path))[0]
            known_encodings.append(encoding)
            labels.append(label)
        else:
            print(f"No faces found in: {image_path}")
    except UnidentifiedImageError as e:
        print(f"Skipping non-image file: {image_path}")
```

```
No faces found in: Images/Image1.png
No faces found in: Images/Image2.png
No faces found in: Images/Image3.png
```

Train a face recognition model

In [ ]:
```python
# Save the face recognition model
face_recognition_model = {
    'known_encodings': known_encodings,
    'labels': labels
}
```

Save the trained model for later use

In [ ]:
```python
joblib.dump(face_recognition_model, "Images/face_recognition_model.pkl")

# Techniques for face identification and verification:
# For face identification and verification, you can use techniques like k-Nearest Neighbors (k-NN) or distance-based algorithms.
```

Out[ ]:
```
['Images/face_recognition_model.pkl']
```

In [ ]:
```python
import face_recognition
```

Load the trained model

In [ ]:
```python
# Load the face recognition model
loaded_model = joblib.load("Images/face_recognition_model.pkl")
```

Identify a face

In [ ]:
```python
def identify_face(input_image):
    try:
        input_encodings = face_recognition.face_encodings(input_image)

        # Check if at least one face is found
        if not input_encodings:
            print("No faces found in the input image.")
            return None

        input_encoding = input_encodings[0]  # Assuming a single face in the input image
```

```
        # Compare the face encoding with the known encodings
        distances = face_recognition.face_distance(loaded_model['known_encodings'], input_encoding)
        min_distance_index = np.argmin(distances)
        identified_label = loaded_model['labels'][min_distance_index]

        return identified_label

    except UnidentifiedImageError as e:
        print(f"Error processing input image: {e}")
        return None
```

Verify a face

In [ ]:
```
def verify_face(input_image, claimed_identity):
    input_encoding = face_recognition.face_encodings(input_image)[0]  # Assuming a single face in the input image

    # Compare the face encoding with the known encodings
    distances = face_recognition.face_distance(loaded_model['known_encodings'], input_encoding)
    min_distance = np.min(distances)

    # Set a threshold for verification
    threshold = 0.6

    if min_distance <= threshold:
        return claimed_identity
    else:
        return "Verification failed"


# Evaluating the performance of the face recognition system, including metrics like accuracy, precision, and recall:
# To evaluate the performance of a face recognition system, you can calculate various metrics:
```

Function to evaluate the face recognition system

In [ ]:
```
# Evaluate the performance of the face recognition system
def evaluate_system():
    true_positives = 0
    true_negatives = 0
    false_positives = 0
    false_negatives = 0
```

```
    for image_path in image_paths:
        try:
            print("Inside Try")
            image = face_recognition.load_image_file(image_path)
            print("1")
            label = os.path.splitext(os.path.basename(image_path))[0]
            print("2")
            # Identify the face
            identified_label = identify_face(image)
            print("identified_label",identified_label)
            # Calculate metrics
            if label == identified_label:
                if label == claimed_identity:
                    true_positives += 1
                else:
                    true_negatives += 1
            else:
                if label == claimed_identity:
                    false_negatives += 1
                else:
                    false_positives += 1
        except UnidentifiedImageError as e:
            print(f"Skipping non-image file: {image_path}")

    # Calculate accuracy, precision, and recall
    accuracy = (true_positives + true_negatives) / len(image_paths)
    precision = true_positives / (true_positives + false_positives)
    recall = true_positives / (true_positives + false_negatives)

    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)

# In this example, the evaluate_system() function compares the identified labels with the true labels and calculates metrics such
```

Note: The specific implementation and choice of metrics may vary based on your requirements and the chosen face recognition library or model.

Complete Code

In [ ]:
```
import os
import face_recognition
import numpy as np
```

```
import joblib
from PIL import UnidentifiedImageError

dataset_path = "Images/"

# Load images and compute face encodings
image_paths = [os.path.join(dataset_path, f) for f in os.listdir(dataset_path) if f.lower().endswith(('.png', '.jpg', '.jpeg', '.
known_encodings = []
labels = []

for image_path in image_paths:
    try:
        image = face_recognition.load_image_file(image_path)
        face_encodings = face_recognition.face_encodings(image)

        # Check if at least one face is found
        if face_encodings:
            encoding = face_encodings[0]  # Assuming a single face in each image
            label = os.path.splitext(os.path.basename(image_path))[0]
            known_encodings.append(encoding)
            labels.append(label)
        else:
            print(f"No faces found in: {image_path}")
    except UnidentifiedImageError as e:
        print(f"Skipping non-image file: {image_path}")

# Save the face recognition model
face_recognition_model = {
    'known_encodings': known_encodings,
    'labels': labels
}

joblib.dump(face_recognition_model, "Images/face_recognition_model.pkl")

# Load the face recognition model
loaded_model = joblib.load("Images/face_recognition_model.pkl")

def identify_face(input_image):
    try:
        print("Running identify_face()")
        input_encodings = face_recognition.face_encodings(input_image)

        # Check if at least one face is found
        if not input_encodings:
```

```
            print("No faces found in the input image.")
            return None

        input_encoding = input_encodings[0]  # Assuming a single face in the input image

        # Compare the face encoding with the known encodings
        distances = face_recognition.face_distance(loaded_model['known_encodings'], input_encoding)
        min_distance_index = np.argmin(distances)
        identified_label = loaded_model['labels'][min_distance_index]

        return identified_label

    except UnidentifiedImageError as e:
        print(f"Error processing input image: {e}")
        return None

def verify_face(input_image, claimed_identity):
    print("Running verify_face()")
    input_encoding = face_recognition.face_encodings(input_image)[0]  # Assuming a single face in the input image

    # Compare the face encoding with the known encodings
    distances = face_recognition.face_distance(loaded_model['known_encodings'], input_encoding)
    min_distance = np.min(distances)

    # Set a threshold for verification
    threshold = 0.6

    if min_distance <= threshold:
        return claimed_identity
    else:
        return "Verification failed"

# Evaluate the performance of the face recognition system
def evaluate_system():
    print("Running evaluate_system()")
    true_positives = 0
    true_negatives = 0
    false_positives = 0
    false_negatives = 0

    for image_path in image_paths:
        try:
            print("Running Loop with Images")
            image = face_recognition.load_image_file(image_path)
```

```
            label = os.path.splitext(os.path.basename(image_path))[0]

            # Identify the face
            identified_label = identify_face(image)
            print("identified_label",identified_label)
            # Calculate metrics
            if label == identified_label:
                if label == claimed_identity:
                    true_positives += 1
                else:
                    true_negatives += 1
            else:
                if label == claimed_identity:
                    false_negatives += 1
                else:
                    false_positives += 1
        except UnidentifiedImageError as e:
            print(f"Skipping non-image file: {image_path}")

    # Calculate accuracy, precision, and recall
    accuracy = (true_positives + true_negatives) / len(image_paths)
    precision = true_positives / (true_positives + false_positives)
    recall = true_positives / (true_positives + false_negatives)

    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)

# Example usage
claimed_identity = "James"
test_image_path = "Images/James.jpg"
test_image = face_recognition.load_image_file(test_image_path)

identified_person = identify_face(test_image)
verification_result = verify_face(test_image, claimed_identity)

print("Identified Person:", identified_person)
print("Verification Result:", verification_result)

# Evaluate the system
evaluate_system()
```

```
No faces found in: Images/Image1.png
No faces found in: Images/Image2.png
No faces found in: Images/Image3.png
Running identify_face()
Running verify_face()
Identified Person: James
Verification Result: James
Running evaluate_system()
Running Loop with Images
Running identify_face()
No faces found in the input image.
identified_label None
Running Loop with Images
Running identify_face()
No faces found in the input image.
identified_label None
Running Loop with Images
Running identify_face()
No faces found in the input image.
identified_label None
Running Loop with Images
Running identify_face()
identified_label Image4
Running Loop with Images
Running identify_face()
identified_label Image5
Running Loop with Images
Running identify_face()
identified_label input_image
Running Loop with Images
Running identify_face()
identified_label James
Running Loop with Images
Running identify_face()
identified_label Output1
Running Loop with Images
Running identify_face()
identified_label James
Accuracy: 0.5555555555555556
Precision: 0.2
Recall: 1.0
```

---

2. Revision of numpy

Refer to Level 1 - Session 1 & 2

3. Revision of Open CV

Refer to Level 1 - Session 3

4. Revision of pandas

Refer to Level 1 - Session 4

---

5. CSV in Data Analysis

What is CSV (Comma-Separated Values)? CSV (Comma-Separated Values) is a plain-text file format commonly used for storing and exchanging tabular data. It uses commas to separate values within each row and newline characters to separate rows. CSV files are simple and widely supported, making them a popular choice for data storage and exchange.

Structure and characteristics of CSV files:

Each row represents a data record, and each column represents a data field. Values within each row are separated by commas. The first row often contains column headers. CSV files are plain text files and can be opened with any text editor. CSV files have a flat structure and do not support nested or hierarchical data. Advantages and common use cases of CSV in data analysis: Advantages:

CSV files are human-readable and platform-independent. They can be easily created, edited, and processed using various software tools. CSV files have a small file size compared to other file formats like Excel. They are widely supported by programming languages and data analysis libraries. Common use cases:

Importing and exporting data between different software applications and databases. Storing and sharing structured data in a simple and portable format. Conducting data analysis and exploratory data analysis (EDA). Creating datasets for machine learning and statistical modeling. Reading CSV files using built-in Python libraries (e.g., csv module, pandas library): Reading CSV using the csv module:

Writing from lists using the csv module:

In [ ]:
```
import csv
import pandas as pd

data = [['Name', 'Age'], ['John', 25], ['Alice', 30], ['Bob', 35]]

# Writing to CSV using the csv module
with open('Images/new1.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(data)

# Writing from a Pandas DataFrame
columns = data[0]
rows = data[1:]

df = pd.DataFrame(rows, columns=columns)
df.to_csv('Images/new2.csv', index=False)
```