**Perform any one:**

* **Mini Project:** Human Face Recognition
* **Mini Project:** Gender and Age Detection: predict if a person is a male or female and also their age.
* **Mini Project:** Colorizing Old B&W Images: color old black and white images to colorful images

The initial lines import necessary libraries. import cv2 imports the OpenCV library, which is a comprehensive library for computer vision tasks, often used here for image manipulation like drawing rectangles and text. import face\_recognition imports the face\_recognition library, which provides tools for recognizing and manipulating faces in images. It's built using dlib's state-of-the-art face recognition technology. import numpy as np imports the NumPy library, essential for numerical operations, especially for handling arrays of face encodings. import os imports the os module, which provides a way of using operating system dependent functionality, such as interacting with the file system. import matplotlib.pyplot as plt imports the pyplot module from the matplotlib library, used here for displaying images. The line %matplotlib inline is a magic command specific to Jupyter notebooks (indicated by the % symbol). It ensures that any plots generated by matplotlib are displayed directly within the notebook output.

The next two lines define string variables for directory names. KNOWN\_FACES\_DIR = "known\_faces" assigns the string "known\_faces" to the variable KNOWN\_FACES\_DIR. This variable will hold the name of the directory where images of known individuals are stored. TEST\_IMAGES\_DIR = "test\_images" similarly assigns "test\_images" to TEST\_IMAGES\_DIR, indicating the directory containing images where face recognition will be performed.

The code then defines a function called load\_known\_faces(). This function is responsible for loading the face encodings and names from the images in the KNOWN\_FACES\_DIR. Inside the function, known\_face\_encodings = [] initializes an empty list to store the numerical representations (encodings) of the known faces. known\_face\_names = [] initializes an empty list to store the names of the known individuals, which will be derived from the filenames of their images. The code then iterates through each filename found within the KNOWN\_FACES\_DIR using for filename in os.listdir(KNOWN\_FACES\_DIR):. An if filename.endswith(('.jpg', '.png')): condition checks if the current file's name ends with either '.jpg' or '.png', ensuring that only image files are processed. If it's an image file, image\_path = os.path.join(KNOWN\_FACES\_DIR, filename) constructs the full path to the image file by joining the KNOWN\_FACES\_DIR and the filename. image = face\_recognition.load\_image\_file(image\_path) uses the face\_recognition library to load the image file into a NumPy array. encoding = face\_recognition.face\_encodings(image) uses the face\_recognition library to compute the face encodings for any faces found in the loaded image. This function returns a list of face encodings. The if len(encoding) > 0: condition checks if at least one face was detected in the image. If a face is found, known\_face\_encodings.append(encoding[0]) appends the first face encoding found in the image to the known\_face\_encodings list (it assumes there's only one main face per known person's image for simplicity). known\_face\_names.append(os.path.splitext(filename)[0]) extracts the filename without the extension (e.g., "elon\_musk" from "elon\_musk.jpg") and appends it to the known\_face\_names list. If no faces are found in an image, the else block executes, printing a message indicating that No faces found in {filename}. Finally, the function returns the known\_face\_encodings list and the known\_face\_names list.

Next, a function called recognize\_faces\_in\_image is defined. This function takes the image\_path of a test image, the known\_face\_encodings, and the known\_face\_names as input. Inside the function, test\_image = face\_recognition.load\_image\_file(image\_path) loads the test image using the face\_recognition library. test\_image\_bgr = cv2.cvtColor(test\_image, cv2.COLOR\_RGB2BGR) converts the image from the RGB color format (which face\_recognition and matplotlib use) to the BGR format, which is commonly used by OpenCV. face\_locations = face\_recognition.face\_locations(test\_image) uses the face\_recognition library to find the coordinates (top, right, bottom, left) of all faces present in the test image. face\_encodings = face\_recognition.face\_encodings(test\_image, face\_locations) computes the face encodings for each face found in the test image, using the face\_locations to focus on the detected faces. The code then iterates through each detected face using a for loop: for (top, right, bottom, left), face\_encoding in zip(face\_locations, face\_encodings):. The zip function allows simultaneous iteration over the face locations and their corresponding encodings. Inside the loop, matches = face\_recognition.compare\_faces(known\_face\_encodings, face\_encoding) compares the current face encoding from the test image with all the known face encodings. It returns a list of boolean values, where True indicates a potential match. name = "Unknown" initializes a variable name to "Unknown", which will be updated if a match is found. face\_distances = face\_recognition.face\_distance(known\_face\_encodings, face\_encoding) calculates the Euclidean distance between the current face encoding and each of the known face encodings. Smaller distances indicate a better match. The if len(face\_distances) > 0: condition ensures that there are known faces to compare against. If there are known faces, best\_match\_index = np.argmin(face\_distances) finds the index of the known face encoding with the smallest distance (the best match). if matches[best\_match\_index]: checks if the best match (based on distance) also corresponds to a True value in the matches list. If it does, it means a sufficiently good match was found, and name = known\_face\_names[best\_match\_index] updates the name variable to the name of the matched known person. After identifying the face (or leaving the name as "Unknown"), the code proceeds to draw a rectangle around the detected face and label it. cv2.rectangle(test\_image\_bgr, (left, top), (right, bottom), (0, 255, 0), 2) draws a green rectangle around the face on the BGR image using the coordinates. cv2.rectangle(test\_image\_bgr, (left, bottom - 35), (right, bottom), (0, 255, 0), cv2.FILLED) draws a filled green rectangle below the face to serve as a background for the name. font = cv2.FONT\_HERSHEY\_DUPLEX defines the font style for the text. cv2.putText(test\_image\_bgr, name, (left + 6, bottom - 6), font, 1.0, (255, 255, 255), 1) writes the recognized name (or "Unknown") onto the filled rectangle below the face in white color. After processing all detected faces, test\_image\_rgb = cv2.cvtColor(test\_image\_bgr, cv2.COLOR\_BGR2RGB) converts the processed image back from BGR to RGB format for display with matplotlib. Finally, the code uses matplotlib to display the processed image: plt.figure(figsize=(10, 6)) creates a new figure with a specified size. plt.imshow(test\_image\_rgb) displays the RGB image. plt.title(f"Processed Image: {os.path.basename(image\_path)}") sets the title of the plot to the name of the processed image file. plt.axis('off') hides the axes of the plot. plt.show() displays the image.

The code then defines a main() function, which orchestrates the face recognition process. os.makedirs(KNOWN\_FACES\_DIR, exist\_ok=True) creates the KNOWN\_FACES\_DIR directory if it doesn't already exist. The exist\_ok=True argument prevents an error if the directory already exists. os.makedirs(TEST\_IMAGES\_DIR, exist\_ok=True) does the same for the TEST\_IMAGES\_DIR. known\_face\_encodings, known\_face\_names = load\_known\_faces() calls the load\_known\_faces() function to retrieve the encodings and names of the known individuals. The if not known\_face\_encodings: condition checks if any known faces were loaded. If not, it prints a message prompting the user to add images to the 'known\_faces' folder and then returns, stopping further execution. If known faces were loaded, the code proceeds to process the test images. It iterates through each filename in the TEST\_IMAGES\_DIR using for filename in os.listdir(TEST\_IMAGES\_DIR):. An if filename.endswith(('.jpg', '.png')): condition ensures only image files are processed. For each test image, image\_path = os.path.join(TEST\_IMAGES\_DIR, filename) constructs the full path, and print(f"Processing {filename}...") prints a message indicating which image is being processed. Finally, recognize\_faces\_in\_image(image\_path, known\_face\_encodings, known\_face\_names) calls the recognize\_faces\_in\_image function to perform face recognition on the current test image using the loaded known faces.

The last line, main(), calls the main() function to start the entire face recognition process.