Lance Mackenzie Go ; Jose Gabriel Salvador ; Phoebe Joyce L. Tan *ITMGT25.03-C*

Introduction and Rationale

We decided to create an image detection program that can be used for businesses to log the attendance of their employees. The program detects the face/image of a person and checks the pre-existing images stored in the designated folder to see if there is a match. It then automatically records the 'time-in' of the employees in a ".csv" file upon the camera successfully detecting a match of the face. The program makes use of neural networks and visual encodings in order to record and match the face. This program will be able to reduce the resources expended in tracking the attendance of employees and help businesses minimize costs.

Pseudocode

In the program, a path was first created to lead to the folder where the images of the people to be detected are kept in. This is then followed by the variable myList which returns a list containing the names of the entries in the directory provided by the folder.

Next, a loop function is created to load all the images in the directory. Here, we extract the names of the persons in the image and calculate the face encoding vector so that we can store the names into a list. The next step is to encode the images and their information into lists, so a function is defined in which we cycle through all the images in the directory, getting the color space of the person using "cv2.cvtColor()", and appending this information into the list returned by the code "face_recognition.face_encodings(img)[0]".

Now that we have the information from the images in the lists, we create another function that would help us automatically put the names of the persons detected in the webcam in the csv file prepared for it. We open the file as "r+" because we hope to not just read it, but edit it as well. To edit the csv, we use the readlines() code to return a list containing each line in the csv as a list item. Here, we append the names of the images into the "list of names" list that is to be added to the csv. In the case that the name of the person is not in the list of names, then it is added to the csv together with the current date and time string.

The next half of the code contains the processes the program undergoes when using the actual webcam, as seen in the initialization of the cv2 VideoCapture class. This helps us be able to capture the video from the camera and turn it into an object. We use a "while" loop to read the video on camera and resize the images on the webcam before comparing it to the images in the people folder. Then, we loop through and get each of the face locations and the encodings in the video and compare these to the encodings of the faces in the picture dataset.

We use the "face_recognition.compare_faces" code to do this–we compare a list of our face encodings against the encoding we have from the camera to see if they match, and this function returns to us a list of True or False values that indicate which among the "current_encoding_list" list gives us a match. We then calculate the facial similarity between the

encoding of the images in the folder and the actual image in the webcam using the “face_recognition.face_distance” line that returns a numpy ndarray with a “distance” that tells us how similar the faces are.

The code is then programmed to return the name of the person that it matches with in the people’s folder if its “distance” provided from face_distance is less than 0.50. If it does not match any identity in the folder, then the program will return “unknown”.

Finally, we create the rectangular box that will appear on the identified person’s face using cv’s cv.rectangle() code, allowing us to manipulate the vertex of the rectangle, its color, thickness, line type, and shift. Then, we use cv2.putText to give the rectangle its label on the screen.