Lab Report for

CE6023 – Computer Vision Systems

Student Name : **Dylan Rodrigues**

Student ID : **24121479**

Revision Timestamp: 03/11/2024 13:19:44

Contents

Introduction 2
Understanding confidence score calculation for Facial Recognition 2
Experiment with Camera Settings 4
Conclusion 6
References..7

Figure 1: My Face along with my Name as a label with a confidence score on the console 4
Figure 2: An increase in Brightness improved the confidence score 6

Introduction

This lab focused on enhancing a facial recognition system on a Raspberry Pi by incorporating confidence score calculations and experimenting with camera settings. Facial recognition is widely used in various fields, from security to personal device unlocking, where accuracy is paramount. A critical aspect of improving the reliability of such systems is evaluating the confidence of each detection. By implementing a confidence score based on Euclidean distance between detected faces and known encodings, the system provides feedback on the likelihood of a correct identification, helping users gauge the accuracy of each recognition event. Additionally, tuning camera parameters like brightness, contrast, sharpness, and dynamic range compression (drc_strength) allows the system to adapt to different environmental conditions, thereby improving detection performance. This lab not only demonstrated the technical aspects of confidence calculation but also underscored the importance of environmental adjustments in real-time computer vision applications.

Through a series of systematic experiments, this lab explored how camera parameter adjustments affected the facial recognition system's performance. By testing one parameter at a time, I could identify optimal settings that produced clearer, more reliable detections. This structured approach helped build an understanding of how different environmental factors can influence a computer vision model's performance, particularly when implemented on resource-constrained hardware like the Raspberry Pi. These experiments provided insights into developing robust and adaptable computer vision systems that can function accurately across various conditions.

Understanding confidence score calculation for Facial Recognition

The objective of this lab activity was to integrate a confidence score calculation into a facial recognition system running on a Raspberry Pi 4 (RPi4) with a connected camera module. This feature was designed to assess the accuracy of detected faces by calculating how close a match is to known faces based on Euclidean distance. Establishing a confidence score adds a layer of interpretability to the model's results, helping to gauge the likelihood that a detected face is correctly classified. This lab session involved modifying an existing Python script to incorporate these calculations and display the scores directly on the video feed.

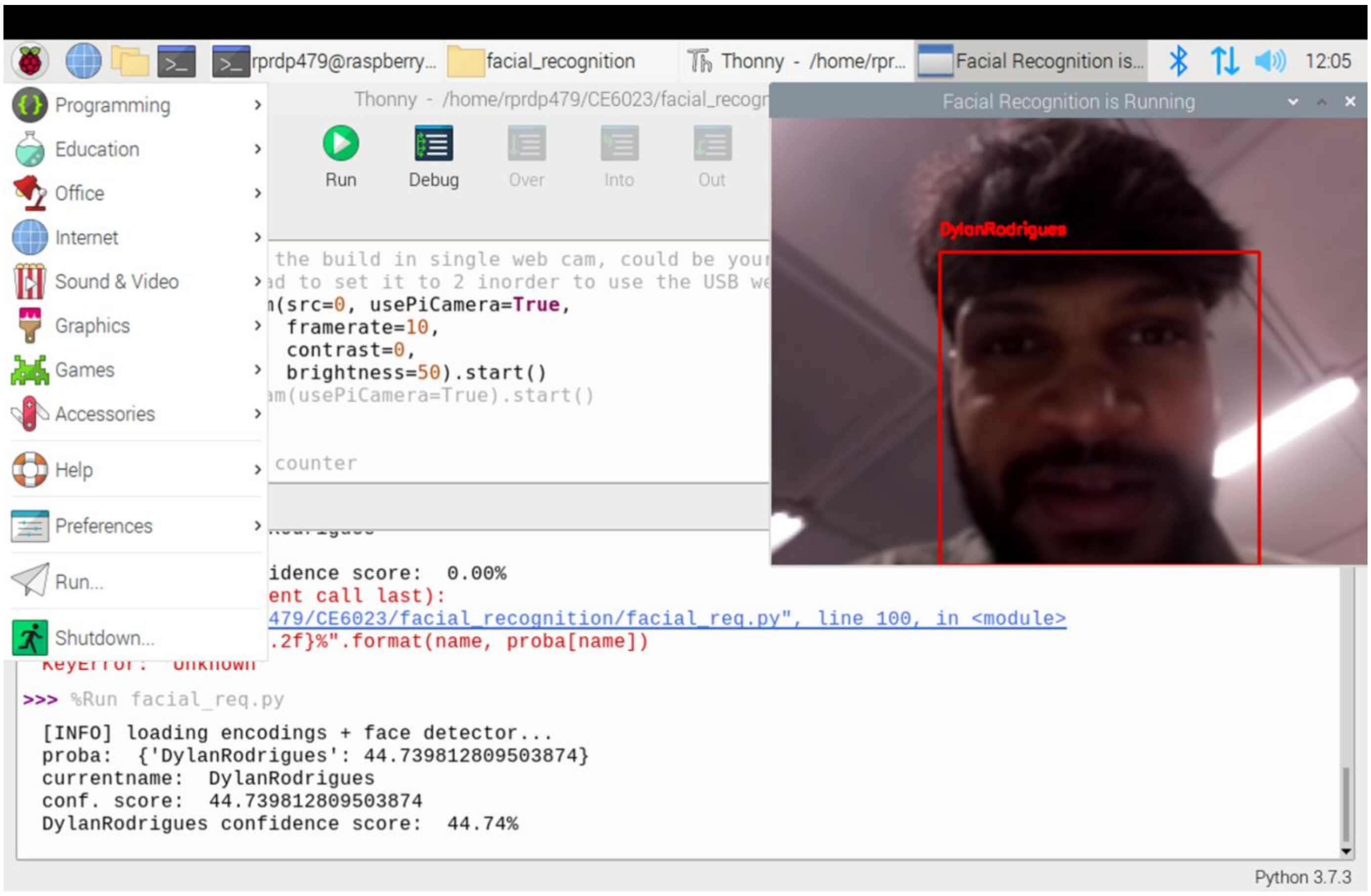
To begin, I powered up both my laptop and the Raspberry Pi 4 setup, connecting them remotely using the Windows Remote Desktop Protocol (xrdp). This connection allowed me to control the Raspberry Pi from my laptop, facilitating easy access to files and script modification. Following the instructions, I logged in as a secondary user with specific credentials from a previous lab session, which helped to reinforce the use of multi-user access on the Raspberry Pi. I then navigated to the `CE6023/facial_recognition` repository within the Raspberry Pi's file manager, which houses the facial recognition code. This familiarized me with the Raspberry Pi's desktop interface and file management, which is essential when handling multiple scripts and dependencies in a remote environment.

The primary script to be modified was `facial_req.py`, the core of the facial recognition functionality. To prepare the script for confidence score calculations, I imported the `statistics` package to leverage additional statistical functions. Modifications to the video stream parameters were also necessary to optimize image quality from the Raspberry Pi camera. Specifically, I adjusted the `usePiCamera` parameter to `True` to activate the Pi camera for video capture and set `contrast` and `brightness` to 0 and 50, respectively. These adjustments ensured better visual clarity and enhanced the model's ability to distinguish faces by fine-tuning the visual feed properties.

The next step involved introducing a confidence score variable. I created a dictionary named `proba` to store the confidence scores for each face the system detected. The confidence score itself was derived from the Euclidean distance between the current frame's face encoding and the known encodings of faces stored by the system. The Euclidean distance serves as a measure of similarity between two points, where a lower value signifies a closer match to a known face and a higher value indicates less similarity. This distance-based confidence score was scaled between 0 and 1, with a score of 0 representing a perfect match and 1 indicating no similarity. To further refine the score, I applied a threshold of 0.5, assigning any distance above this value a score of 1 to reduce confidence for less accurate matches. This approach added robustness to the recognition process by helping to distinguish strong matches from weaker ones.

With the confidence score logic in place, I proceeded to enhance the script's real-time display of detection results. I added a new variable, `text`, to format the display output so that each detected face would show both the identified name and the associated confidence score. To improve visibility, I updated the color scheme in the OpenCV bounding box and text display functions: the bounding box around detected faces was changed to red `(0, 0, 255)`, while the text was also modified to appear in red, ensuring that both the box and confidence information were highly visible on screen. Additionally, I reduced the text size to fit well within the bounding box and avoid visual clutter, which made the display suitable for real-time monitoring.

Finally, I tested the modified facial recognition system, observing how the confidence score influenced the detection output. Running the script on the Raspberry Pi, I could see my face detected and labeled with a confidence score (as shown in Figure 1: My Face along with my Name as a label with a confidence score on the console), confirming that the calculations were functioning correctly. This enhanced feedback was useful, especially in observing how changes in facial position or lighting affected the confidence score. By capturing a screenshot of the running system with my face detected and the confidence score displayed, I documented the system's performance and accuracy. This exercise underscored the value of confidence scores in practical applications, as they provide users with a quick visual cue regarding the reliability of the facial recognition system's output. This lab was a valuable experience in implementing confidence-based metrics for real-time computer vision tasks.



```
the build in single web cam, could be your
ad to set it to 2 inorder to use the USB we
n(src=0, usePiCamera=True,
   framerate=10,
   contrast=0,
   brightness=50).start()
am(usePiCamera=True).start()

counter
```

```
idence score:  0.00%
ent call last):
479/CE6023/facial_recognition/facial_req.py", line 100, in <module>
.2f}%".format(name, proba[name])
KeyError: unknown
>>> %Run facial_req.py
  [INFO] loading encodings + face detector...
  proba:  {'DylanRodrigues': 44.739812809503874}
  currentname:  DylanRodrigues
  conf. score:  44.739812809503874
  DylanRodrigues confidence score:  44.74%
```

Figure 1: My Face along with my Name as a label with a confidence score on the console

Experiment with Camera Settings

After implementing the confidence score calculation, the next step was to explore how adjusting the camera settings could impact the facial recognition system's accuracy. For this experiment, I focused on systematically varying parameters in the `VideoStream` function, such as brightness, contrast, sharpness, and `drc_strength`. By modifying these settings one at a time, I could better observe the effect each adjustment had on the system's performance, particularly in terms of detection reliability and confidence scores.

To begin, I experimented with the brightness setting, which was initially set to a default value of 50. I incrementally increased brightness in steps of 10, running the facial recognition model with each adjustment. At higher brightness values, I noticed that the system initially showed an increase in confidence scores, as the enhanced lighting improved the visibility of facial features (as illustrated in

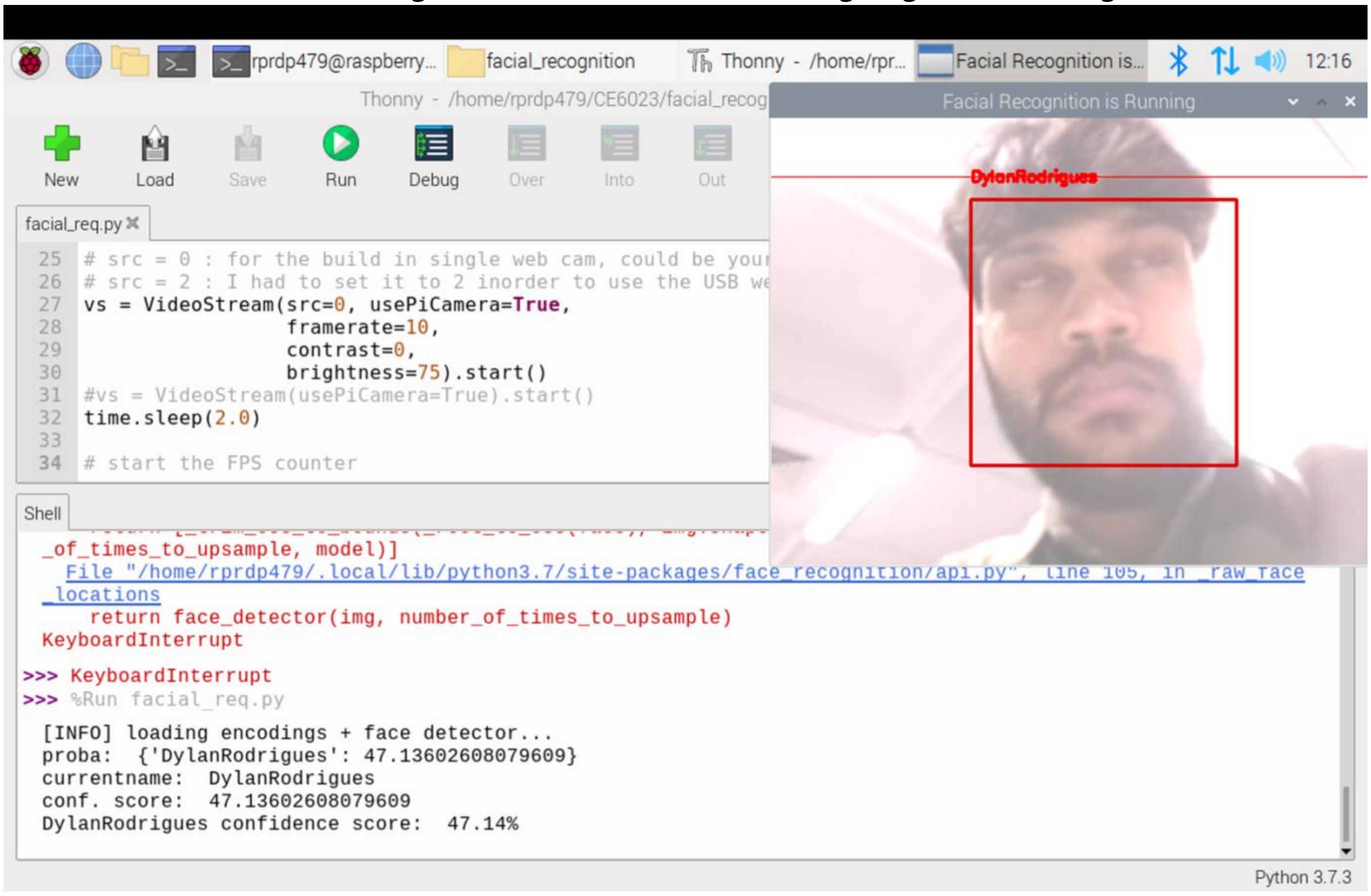
Figure 2: An increase in Brightness improved the confidence score). However, as brightness continued to increase, overexposure reduced the quality of these features. At a brightness value of around 80, the model struggled to detect faces reliably, with confidence scores dropping significantly. This demonstrated that while a moderate increase in brightness could help in low-light conditions, excessive brightness negatively impacted detection by washing out key facial features.

I then focused on the contrast parameter, adjusting it across a range from low (10) to high (90) values. Increasing contrast initially improved the system's ability to differentiate facial features, which led to higher confidence scores. However, as contrast values approached the higher end of the spectrum, the model once again showed a decline in detection performance. This effect likely stemmed from exaggerated differences in light and dark areas, which caused the HOG-based detector to misinterpret certain facial regions. Through this experiment, I found that a moderate contrast setting (around 40-60) yielded the best detection accuracy and confidence scores.

Next, I tested the impact of the sharpness setting, starting from a base value of 0 and gradually reducing it to -100, then increasing it to positive values. With sharpness set to lower levels (below -50), the images appeared softer, causing the HOG detector to perform poorly, as essential edge details became blurred. On the other hand, increasing sharpness enhanced the clarity of edges and features, slightly improving the model's accuracy. However, excessive sharpness also led to some inconsistencies, as overly defined edges occasionally caused false positives. Based on these observations, I determined that keeping sharpness close to the base value of 0 worked best for this particular setup.

Lastly, I experimented with the `drc_strength` parameter, which adjusts dynamic range compression and is similar to gamma correction. Increasing `drc_strength` helped balance lighting across the image, making it particularly useful in conditions with uneven lighting. By looping through different values of `drc_strength`, I observed that moderate levels (around 2) optimized the model's performance, providing a balanced exposure that retained facial details across varying lighting intensities.

Overall, this experiment highlighted the importance of tuning camera settings to achieve optimal performance in facial recognition tasks. By adjusting brightness, contrast, sharpness, and `drc_strength`, I was able to identify parameter ranges that produced the highest confidence scores, suggesting these settings help create an environment well-suited for accurate and reliable detection. This exploration underscores the significance of environmental adjustments in real-time computer vision applications, especially when deploying systems in diverse lighting conditions.



```
# src = 0 : for the build in single web cam, could be your
# src = 2 : I had to set it to 2 inorder to use the USB we
vs = VideoStream(src=0, usePiCamera=True,
                 framerate=10,
                 contrast=0,
                 brightness=75).start()
#vs = VideoStream(usePiCamera=True).start()
time.sleep(2.0)

# start the FPS counter
```

```
  _of_times_to_upsample, model)]
    File "/home/rprdp479/.local/lib/python3.7/site-packages/face_recognition/api.py", line 105, in _raw_face
  _locations
      return face_detector(img, number_of_times_to_upsample)
  KeyboardInterrupt

>>> KeyboardInterrupt
>>> %Run facial_req.py

  [INFO] loading encodings + face detector...
  proba:  {'DylanRodrigues': 47.13602608079609}
  currentname:  DylanRodrigues
  conf. score:  47.13602608079609
  DylanRodrigues confidence score:  47.14%
```

Figure 2: An increase in Brightness improved the confidence score

Conclusion

In this lab, I successfully integrated a confidence scoring mechanism into a facial recognition system and investigated the impact of camera settings on detection performance. The confidence score, calculated based on Euclidean distance, offered valuable insight into the model's reliability by providing a quantifiable measure of detection accuracy. This metric proved useful in real-time applications, giving immediate feedback on whether detected faces matched known encodings with high or low confidence. Experimenting with camera settings, including brightness, contrast, sharpness, and drc_strength, demonstrated the significant role environmental conditions play in facial recognition performance. Each parameter had an optimal range that improved detection accuracy, but excessive adjustments often resulted in degraded performance.

The experiments underscored the need for fine-tuning when deploying computer vision systems in varying environments. By identifying the best configuration settings, I optimized the system for accurate face detection, particularly on a platform with limited processing power like the Raspberry Pi. This lab provided practical experience in configuring camera-based recognition systems and reinforced the importance of adaptability in computer vision applications. The methods used and insights gained could be beneficial for future projects where accurate, real-time facial recognition is essential, and adapting to different lighting and environmental conditions is required.

References

Vijayabhaskar J. (2020, February 10). *A Gentle Introduction into the Histogram of Oriented Gradients*. Retrieved from Analytics Vidhya: https://medium.com/analytics-vidhya/a-gentle-introduction-into-the-histogram-of-oriented-gradients-fdee9ed8f2aa