

Course Code: AER 850

Course Title: Introduction to Machine Learning

Semester: F2024

Instructor: Dr. Reza Faigehi

Submission: Project 3


Due Date: Sunday, December 14th, 2024

Title: Project 3

Section Number: 02

Submission Date: Sunday, December 14th, 2024

Submission By: Arjun Tripathi

Name	Student Number (XXXX99999)	Signature
Arjun Tripathi	XXXX21964	

By signing the above you attest that you have contributed to this submission and confirm that all work you contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, and "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at www.torontomu.ca/senate/policies/

Table of Contents

Table of Contents.....	2
Introduction.....	3
Object Masking - OpenCV.....	4
Component Detection - YOLOv8 Training & Evaluation.....	7
Conclusion.....	13
Appendix A : Github Repository Link.....	14

Introduction

This project aims to automate the identification and placement of components on a printed circuit board using machine learning and computer vision. There are two parts to this project: image processing through the use of OpenCV and component detection through YOLOv8.

Object Masking - OpenCV

The purpose of object masking through OpenCV was to isolate the motherboard given in the original picture, seen in Figure 1.0. The final image that was needed had to be solely the PCB with all its components and the background of the image removed. Three packages were used to complete this task: OpenCV, Matplotlib and Numpy.



Figure 1.0: Original Image of Motherboard

The process was initiated by loading the original image into the code through the `cv2.imread` function from OpenCV. The image was then rotated 90 degrees clockwise to ensure the correct orientation of the PCB. Then, the image was converted to grayscale to better define the edges and simplify the complexity. Thresholding techniques were then used on the grayscale image to segment based on pixel intensity, separating the object from its background. The following parameters were used for the thresholding technique,

- Threshold Value = 100
- Maximum Value = 255
- Technique = `cv2.THRESH_BINARY`

This produced a binary image where significant features of only the PCB were highlighted. After this, edge detection and image dilation techniques were implemented, which are portrayed by Figure 2.0. Two techniques were used for edge detection, Canny corner detectors and contour detection. The following parameters were used for both techniques:

- Canny:
 - Lower threshold = 50
 - Upper threshold = 150
- Contours:
 - Mode: `cv2.RETR_EXTERNAL`
 - Algorithm used: `cv2.CHAIN_APPROX_SIMPLE`

- Contour Index = -1
- Color = 255
- Thickness = cv2.FILLED
- Largest Contour was selected.

In regards to image dilation, a rectangular 9x9 kernel was used to dilate the image and two iterations were used.

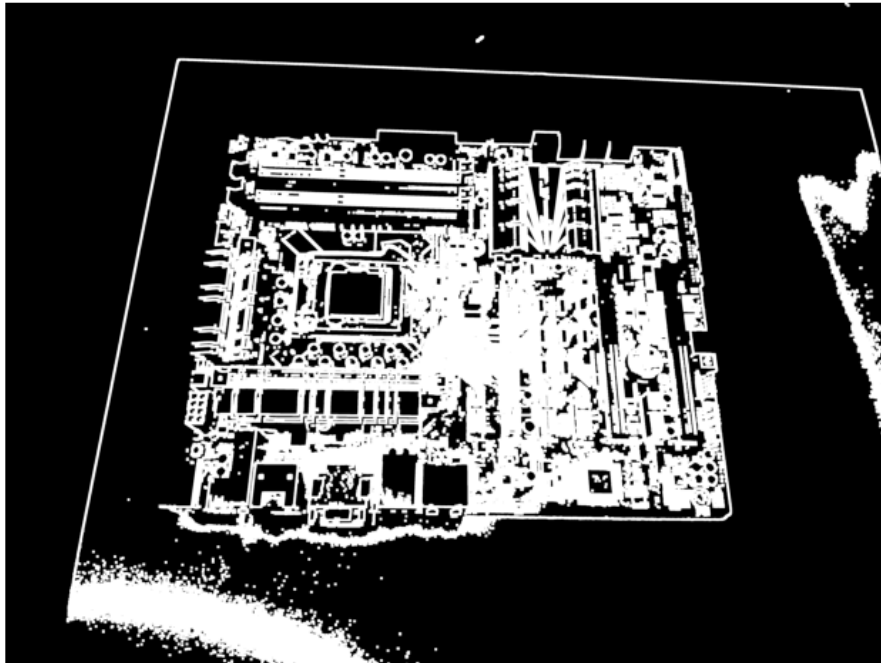


Figure 2.0: Edge Detection of Motherboard Image

Before the contour was applied, a mask for the grayscale image was created using the numpy function `np.zeros_like`, seen in Figure 3.0.



Figure 3.0: Mask Image of Motherboard

Lastly, the contour was applied on to the mask and the mask was applied onto the image through the `cv2.bitwise_and` function. Before saving and showcasing the image, a gaussian blur with a kernel size of 5x5 was implemented onto the image to smooth the extracted image. The final extracted image can be visualized in Figure 4.0 below. Due to overlapping components and imbalances in lighting, there were imprecise extractions in some edges such as the bottom left corner of the image. This can be improved by using a more complex thresholding technique such as adaptive thresholding or by using a gaussian blur in the original image before any edge detection techniques are applied. Furthermore, edge detection parameters can also be changed to tighten the threshold and address the lighting variability. All pictures can be found in the github repository linked in Appendix A.



Figure 4.0: Final Extracted Image

Component Detection - YOLOv8 Training & Evaluation

The YOLOv8 nano model was trained and fine-tuned to detect PCB components accurately. The Ultralytics library was used to complete this task. The dataset given was already preformatted for this model. Below are the key parameters used to fine tune the model:

- Hyperparameters:
 - Epochs = 117
 - Batch Size = 7
 - Image Size = 900
- Model was saved for further use in the Google Drive.
- Model was named, PCB Component Detection Model.

To improve this model, the hyperparameters could be further changed such as increasing the Epochs to 150 or changing the batch size and image size to a higher instance. However, this was not possible in this situation due to the lack of processing power in Google Colab.

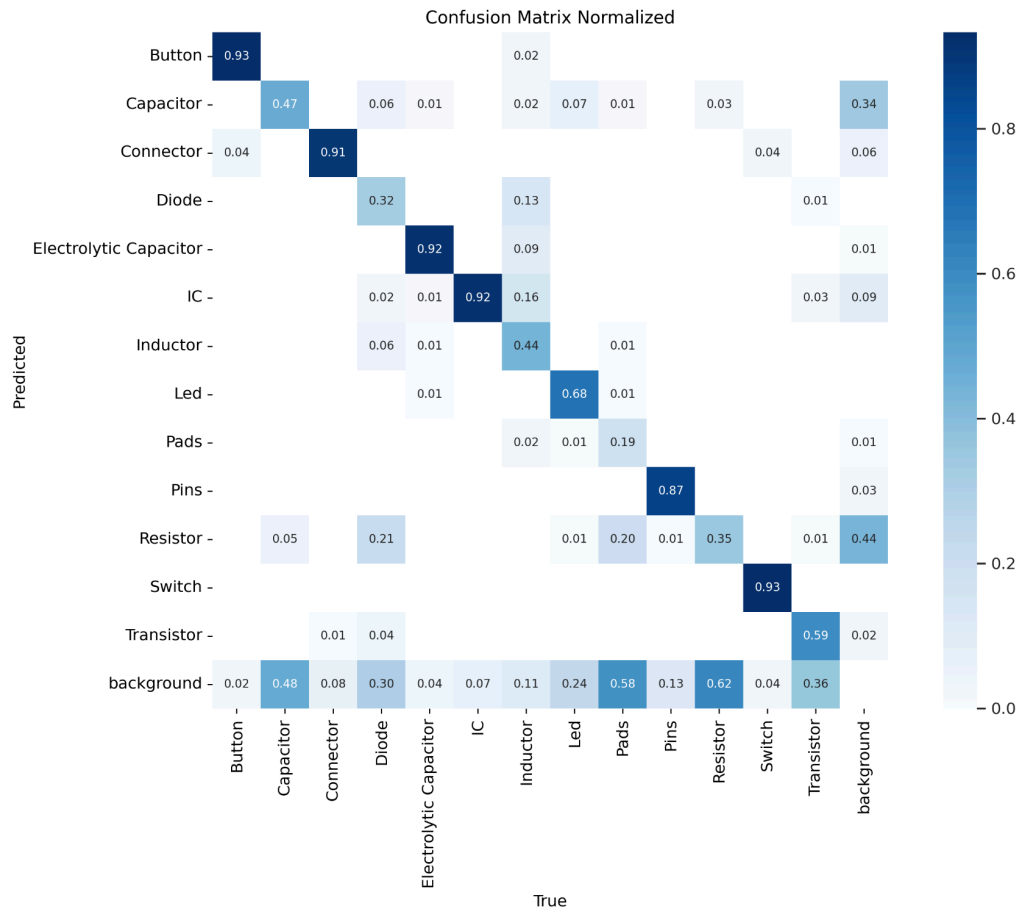


Figure 4.0: Normalized Confusion Matrix

In Figure 4.0, one can see the normalized confusion matrix which shows the correlation of all 13 component classes. Most of the components showed little to no correlation however, there was a general correlation between these classes:

- Background & Components
- Inductor & IC/Diode
- Diode & Resistor
- Pads & Resistor

Furthermore, the Precision-Recall curve shown in Figure 5.0, allowed the observation of the balance between precision and recall between all classes (components).

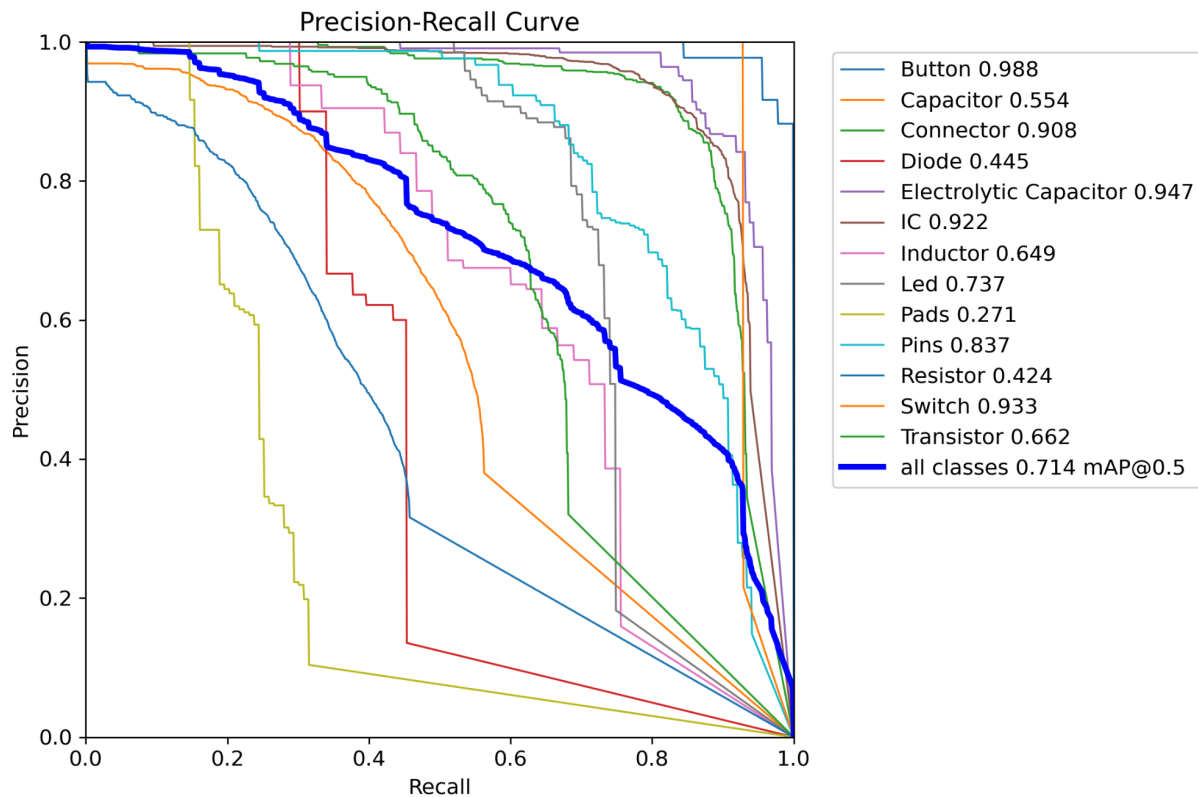


Figure 5.0: Precision-Recall Curve

As observed in Figure 5.0, the model produced a strong balance between both precision and recall amongst all classes, specifically in the recall region of 0.5 to 0.8 where the precision is on an average above 80% for most components. There were some components that did not do so well such as the pads or the resistor and diode, however components like the button showcases an almost perfect precision-recall curve. Each component decreased in precision at high recall levels, this suggests that the model prioritized true positives, with the minimal introduction of false positives. There is definitely room for improvement as many of the curves for the components were not ideal like the button's curve. These improvements can be made by additional training data and time.

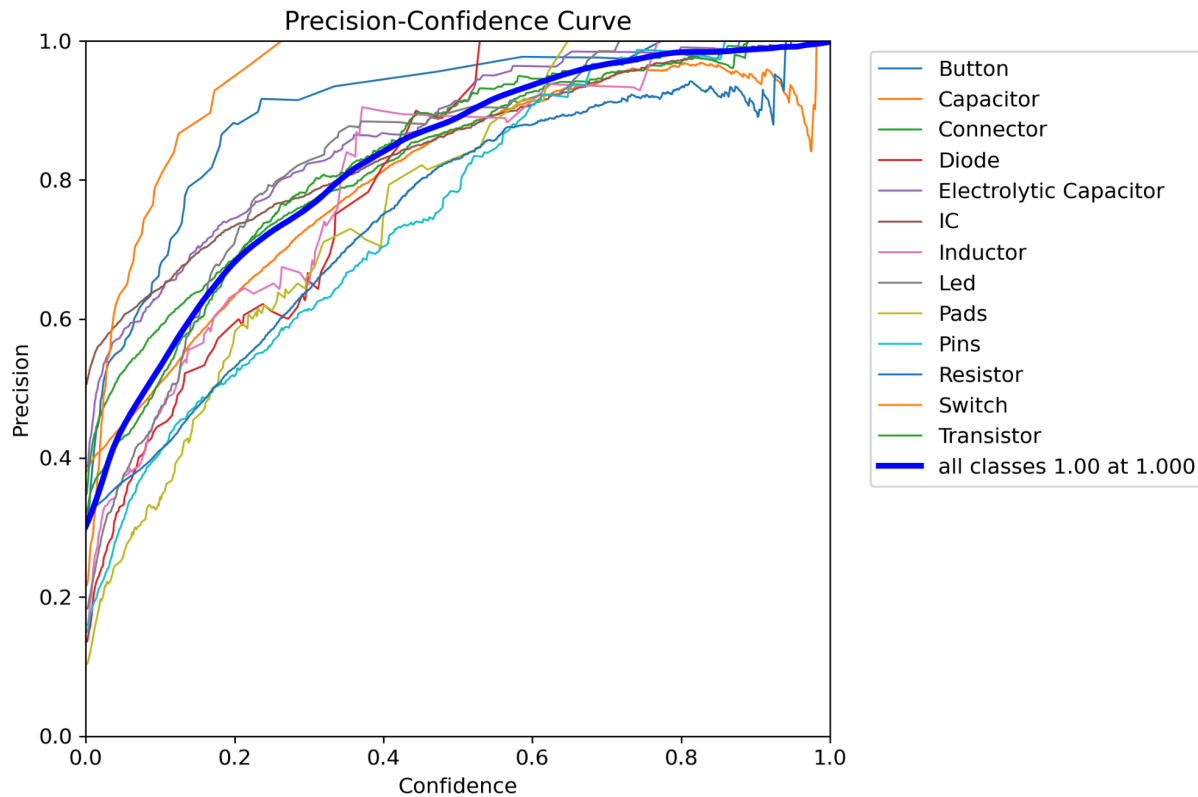


Figure 6.0: Precision-Confidence Curve

Figure 6.0 illustrates the Precision-Confidence curve of all the components in the model. This curve portrays the precision of each classifying each component at various confidence thresholds. As seen from the curves, the precision increased as the confidence threshold was increased, specifically above a 0.6 confidence threshold, the precision is consistently high. There's a great decline below a confidence threshold of 0.6, suggesting that there was noise that may have affected the model.

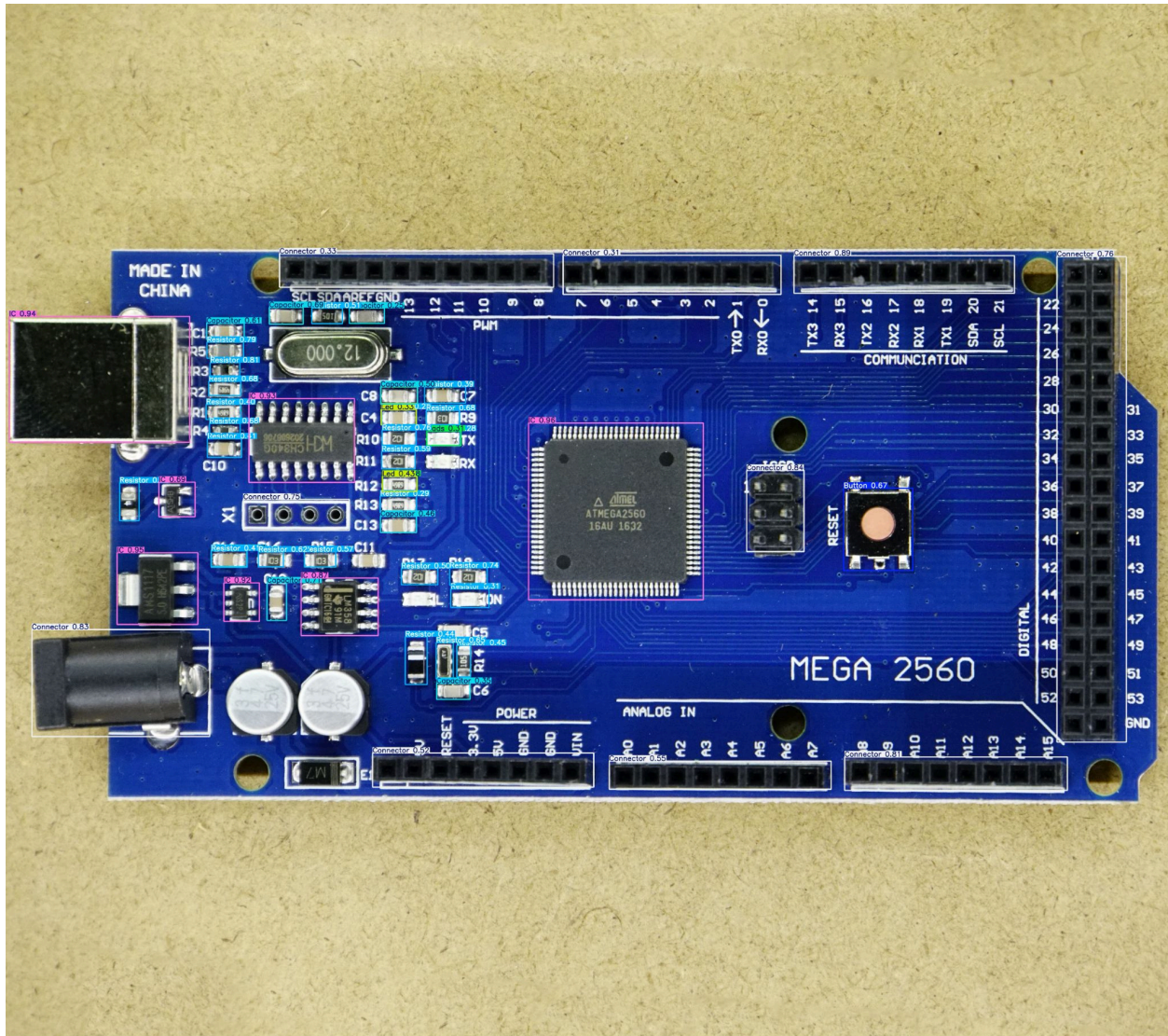


Figure 7.0: Arduino Mega Evaluation Image

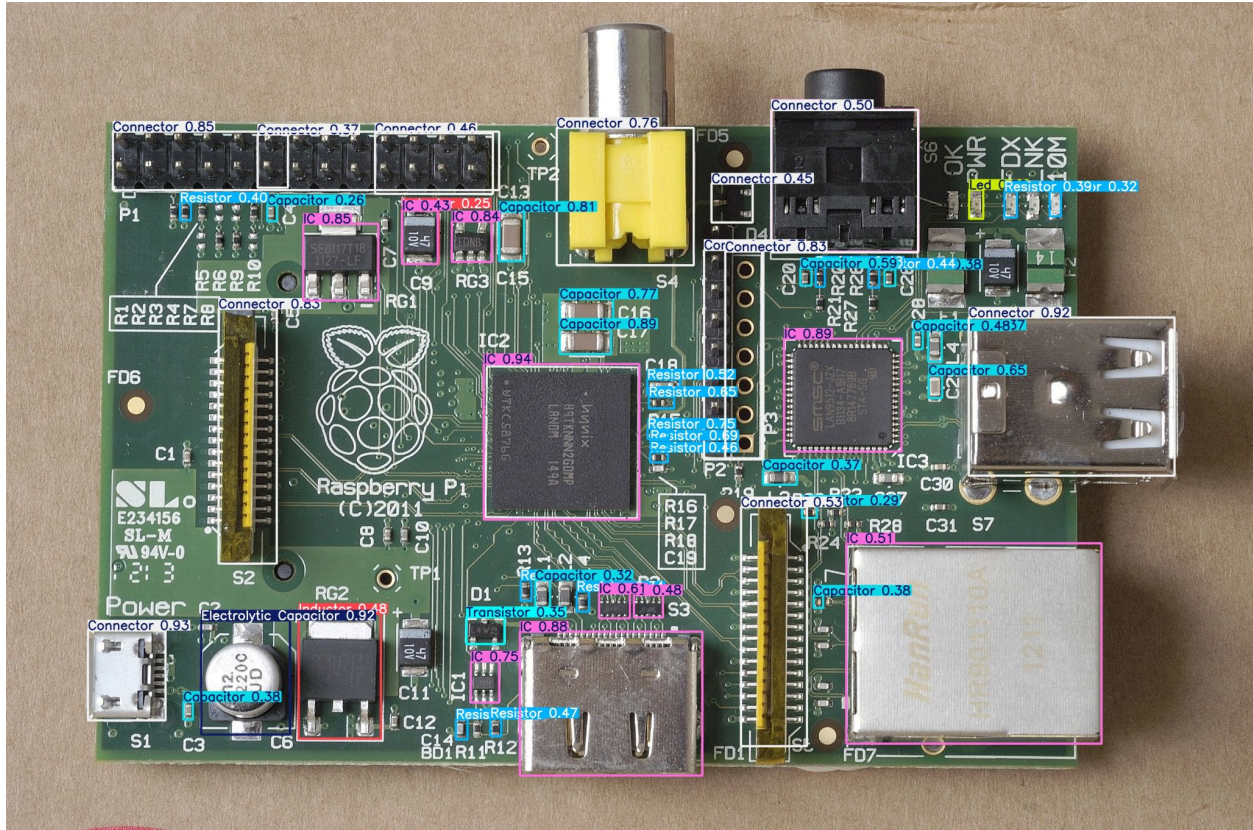


Figure 9.0: Raspberry Pi Evaluation Image

Figure 7.0 , 8.0, & 9.0 depict the evaluated images of the Arduino Mega, Arduino Uno & Raspberry Pi PCBs, respectively. The detection of each component may be hard to see as the line width was decreased for orgranziation purposes. A clear image of each PCB can be seen in the github repository linked in Appendix A. These images were evaluated using the following parameters:

- Image Size: 928
- Confidence Threshold = 0.25
- Line Width = 2/3

Conclusion

This project successfully introduces image processing and detection through OpenCV and YOLOv8. Further improvements can be made by increasing the number of epochs, the batch size and the image size, however, the limit on resources restricted these improvements.

Appendix A : Github Repository Link

Link to GITHUB:

<https://github.com/Arjunt10/AER850Project3.git>

Link to GDrive:

 AER850 Project 3 Files