**Semester, Year:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Semester, Year:** | ✅ Fall | ⬜ Winter | ⬜ Spring/Summer | 2023 |
| **Course Code:** | AER 850 | | | |
| **Course Title:** | Introduction to Machine Learning | | | |
| **Section Number:** | 01 | | | |
| **Instructor:** | Reza Faieghi | | | |
| **Submission:** | ⬜ Assignment ⬜ Lab Report ✅ Project Report ⬜ Thesis  ⬜ Other: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | |
| **Due Date:** | December 17, 2023 | | | |

Project 3 Report

December 17, 2023

|  |  |  |
| --- | --- | --- |
| **Authors/Contributors:** | **Student Number (XXXX12345):** | **Signature\*:** |
| Kotasthane, Gaurang | 500922614 | G.K. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

*\*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.ryerson.ca/academicintegrity/students/ryersons-academic-integrity-policy-policy-60/*](https://www.ryerson.ca/academicintegrity/students/ryersons-academic-integrity-policy-policy-60/)

Table of Contents

[Discussion 3](#_Toc153113482)

[Object Masking 3](#_Toc153113483)

[Thresholding 3](#_Toc153113484)

[Method 3](#_Toc153113485)

[Results 4](#_Toc153113486)

[YOLO v8 Training & Evaluation 5](#_Toc153113487)

[Model Training Process 5](#_Toc153113488)

[Model 1 5](#_Toc153113489)

[Architecture 5](#_Toc153113490)

[Results – Model 1 6](#_Toc153113491)

[Evaluation – Model 1 9](#_Toc153113492)

[Model 2 11](#_Toc153113493)

[Architecture 11](#_Toc153113494)

[Results – Model 2 11](#_Toc153113495)

[Evaluation – Model 2 14](#_Toc153113496)

[Conclusion 16](#_Toc153113497)

# Discussion

## Object Masking

### Thresholding

#### Method

Image thresholding can be done using a few different methods. These methods are readily available to be implemented using the OpenCV library. In this project the following threshold functions were implemented

1. Threshold Binary
2. Threshold Binary Inverse
3. Threshold Trunc
4. Threshold to zero
5. Threshold to zero inverse

All these functions work on a simple principle, the look at the no of pixels at a particular location in the image and then depending on the threshold that was set, it assigns a value of intensity for that location in the image. The report looks at 5 different methods to choose the optimal thresholding function for this project.

The threshold value for the functions was chosen based on a histogram of the intensity and then we can look at different peaks and choose a threshold in between those peaks to effectively separate the feature that needs to be extracted from the image. As illustrated below, a threshold of 60 was chosen for this project based on the histogram.

A graph of a line graph

Description automatically generated with medium confidence

##### Figure #: Histogram

#### Results

A collage of images of a computer chip

Description automatically generated

##### Figure #: Plot of different thresholding functions and their results

By analyzing the plot, it is apparent that the binaryInv function works the best to separate the motherboard from its background.

## Edge Detection

## YOLO v8 Training & Evaluation

### Model Training Process

In this project the model was trained on a custom dataset that was provided and using YOLOv8. For the training I trained two models and I’ll be sharing the thought process and the results below.

As highlighted in the project, the model can only be tuned on three hyperparameters, namely, number of epochs, batch size and image size. The higher the value of these parameters results in higher accuracy and better performing model.

To train, Google Collab was used since it was impossible to train it locally.]

Before going into the architecture and the results of the models, it is important to highlight what I tried and didn’t seem to work.

1. High batch size: Since we are dependent on the virtual ram access provided by Google Collab. The default batch size for the ‘batch’ argument in the training of YOLO is 16. Using this value, the program was always stopped since Google Collab only gives access to 15 GB of VRAM while a batch size of 16 requires much more VRAM than that. Through lots of trial and error, the maximum batch size was determined to be 4. The VRAM used for this was about 10-14 GB which was withing the limits of Google Collab.

Please note that I have included my findings for **two** (**2)** models. This report also has results and evaluations for both the models.

### Model 1

#### Architecture

!yolo task=detect mode=train model=yolov8m.pt data=data.yaml epochs=100 imgsz=1216 plots=True batch=2

Epochs = 100

Imgsz= 1216

Batch=2

#### Results – Model 1

A screenshot of a computer

Description automatically generated

##### Figure 1. Normalized Confusion Matrix (Model 1)

The main diagonal represents the proportion of correct predictions for each class (True positives). It can be inferred from this there are some classes that are misclassified a lot more times than others. For example, ‘Pads’ class has only a 0.58 value on that diagonal which means its only predicted 58% of times which is not satisfactory. The model can be improved on the following classes ‘Pads’, ‘Resistor, and identifying ‘backgrounds.

A graph of different colored lines

Description automatically generated

##### Figure 2: Precision Confidence Curve (Model 1)

The dark blue line indicates that classification of all classes reaches a maximum of 1.0 with 97.5% confidence which seems to be a pretty good result. If we look at different classes, for example classes such as ‘Pads’ have lower precision at lower confidence levels which suggest lower reliability which is something that needs to improve with Model 2.

A diagram of different colored lines

Description automatically generated

##### Precision-Recall Curve (Model 1)

Each curve represents a trade-off between precision and recall for a given class. In terms of overall performance, the dark blue line suggests the mean average of 0.907 at an IoU of 0.5 which suggests that this model is performing pretty good.

#### Evaluation – Model 1

A blue circuit board with wires and wires

Description automatically generated

##### Figure #: IMG: ardmega evaluation (Model 1)

Note: Missed a resistor and capacitor on mid-left of the board. Otherwise, the model performed pretty good.

A close-up of a circuit board

Description automatically generated

##### Figure #: IMG: arduno evaluation (Model 1)

Note: Model seems to have identified everything correctly

A close-up of a circuit board

Description automatically generated

Figure #: IMG: rasppi evaluation (Model 1)

Note: Missed a Connector, capacitor on the mid-left section. Rest of the classes were identified

### Model 2

#### Architecture

yolo task=detect mode=train model=runs/detect/train/weights/last.pt data=data.yaml epochs=200 imgsz=1216 plots=True batch=4 save\_period=10 resume=True

Epochs = 200

Imgsz= 1216

Batch=4

#### Results – Model 2

A screenshot of a computer

Description automatically generated

##### Figure #. Normalized Confusion Matrix (Model 2)

A graph of different colored lines

Description automatically generated

##### Figure 2: Precision Confidence Curve (Model 2)

A graph of different colored lines

Description automatically generated

##### Precision-Recall Curve (Model 2)

#### Evaluation – Model 2

A blue circuit board with wires and wires

Description automatically generated

##### Figure #: IMG: ardmega evaluation (Model 2)

Note: Some of the predicted inaccuracies include the same resistor and capacitor in the mid-left section, the rest of the items have been classified correctly and with a higher confidence level

A close-up of a circuit board

Description automatically generated

Note: Model seemed to have missed the connector on the top right, bottom-mid of the board and the buttons on the top left. For this board the model seems to have performed worse than model 1. Although in the results provided on D2L, the button on the top left is misclassified as a capacitor so small errors are to be expected.

##### Figure #: IMG: arduno evaluation (Model 2)

A close-up of a circuit board

Description automatically generated

##### Figure #: IMG: rasppi evaluation (Model 2)

Note: Model seems to have missed the connector on the top-mid section of the board, rest of the items have been identified correctly

### Conclusion

In terms of the model performance both models seemed to have performed exceptionally well, as apparent from the confusion matrix as well the P-C and P-R curves.

The model can be improved by increasing the batch size and the number of epochs and perhaps object masking since one of the biggest misclassifications was the ‘background’ class.

All the results can be found in the GitHub repository that was shared for authenticity.