Toronto Metropolitan University

Department of Aerospace Engineering
Faculty of Engineering & Architectural Science

Semester (Term, Year)	F2025
Course Code	AER850
Course Section	02
Course Title	Introduction to Machine Learning
Course Instructor	Dr. Reza Faieghi
Submission	3
Submission No.	3
Submission Due Date	December 1, 2025
Title	Project 3 - LOAV-PCB - Kai Stewart
Submission Date	November 30, 2025

Submission by (Name):	**Student ID (XXXX1234)**	**Signature**
Kai Stewart	**29849**	

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 Academic Integrity Policy 60, which can be found at www.torontomu.ca/senate/policies/ *Aerospace Assignment Cover as of May 2022*

1.0 - Introduction

Project 3 of AER850 challenges students to utilize a pretrained nano version of the YOLO (You Only Look Once) model to train and evaluate an object detection model for use on PCBs (printed circuit boards) for identification of components. Students were also challenged with creating an object masking pipeline given an image of a motherboard, and using OpenCV() to successfully extract the motherboard from the background. This project has been named LOAV-PCB (Look Once and Verify PCB), and the project can be found in full on Github in the link below.

Github - Akkoxs - LOAV-PCB

Thankfully, we are working with better hardware this time around and can utilize a powerful GPU using CUDA drivers.

GPU: NVIDIA RTX 4070 Super 12GB GDDR6X VRAM

Models Trained:

1. LOAV-PCB-v0
    a. Epochs = 5, imgsz =900

2. LOAV-PCB-v1
    a. Epochs = 200, imgsz =900

3. LOAV-PCB-v2
    a. Epochs = 200, imgsz =1140

2.0 - Object Masking

The object masking part of this project focused on an image of a Motherboard which was provided with an example of what the final masked image should look like. This example can be seen below in figure 1.

Figure 1: Object Masking EXAMPLE

The object masking pipeline created consists of 6 main steps, which will all be explained in detail below. .

1. Greyscaling
2. Adaptive Thresholding
3. Contours Detection
4. Masking
5. Morphological Transform
6. Binary Thresholding

2.1 - Greyscaling

The first step of any image preprocessing pipeline in the realm of machine learning is usually to greyscale it and remove the “noise” of colour from the picture, this can allow the model to scale every pixel from 0 (black) to 255 (white). This is applied to the motherboard image.

2.2 - Adaptive Thresholding

Classical, binary thresholding takes a grayscale image as an input and using a particular threshold value as a means of decision, converts every pixel into one of two binary categories, white or black. Since our motherboard image has some lighting gradients on it, doing binary thresholding could take some time to get perfect, as we would have to experiment with many different threshold values. Thus, adaptive thresholding was chosen to move forward.

Adaptive thresholding calculates a different threshold for every pixel based on its immediately surrounding pixels, thus it allows the thresholding algorithm to tailor itself to local lighting conditions. The parameters for this operation that were selected and can be fine tuned on this are as follows:

- maxValue = 255
- adaptiveMethod = Gaussian C
- thresholdType = Inverted Binary
- blockSize = 1401
- C = 5

maxValue is of course 255 because we know that since our original image consisted of RGB coloured values, the max value can be 255, as per that scale.

The Gaussian C method as opposed to the Mean C method applies weights onto the surrounding pixels of the target pixel and makes it such that further pixels matter less and closer pixels matter more to the calculation of the adaptive threshold. This is better to work around illumination gradients such as the one present on the motherboard as well as for tiny details which this PCB is full of.

The Inverted Binary type is simply just to invert the scales 0-255, such that we can have our edges be 0 (black) or 255 (white), this was done for a future operation.

blockSize being 1401 is extremely large, but experimentally, was found to provide the best overall image. This parameter is the width and height of the window which is used to compute local threshold values for each pixel, and since the Motherboard image in total is only 4344 x 5792 pixels, this accounted for approximately a quarter of the image at a time. This was great for getting rid of the shadows around the edge of the motherboard, which was necessary to get a clean image in the end.

C parameter is a constant which is subtracted from the blocks average, a bias that can be used to tune the threshold up or down. This was set to 5 arbitrarily based on iteration and how the end results looked.

2.3 - Contour Detection

Please note that corner detection methods such as Canny or Harris were not used because they were found experimentally to detract from the overall image quality. Contour detection was used however with success. A contour is defined as a ‘border’ around an artifact in the image, this is marked by a sudden change in pixel colour, which is how this algorithm recognizes contours. In this case, we know that if we apply contour detection, the largest contour in this image should be the motherboard itself. Thus, we apply contour detection, which returns to us a list of contours ordered from largest to smallest, and we take the first element of this list and save it as our selected contour.

2.4 - Masking

Masking is the process of making a ‘cutout’ so to speak of the part of the image we want, and then using it to overlay the original image so we can get rid of the background and only get what we want. In this case, masking is done by creating a totally black image, and then cutting out the shape of our motherboard using the largest contour from the previous step.

2.5 - Morphological Transform

Next, we define a new kernel of 50x50 pixels and display the motherboard image in terms of this kernel. This removes a bit of small noise on the outskirts of the image. This kernel size was selected experimentally because it was found to not detract from the motherboard image and its components.

2.6 - Binary Thresholding

One last binary thresholding operation is done to smooth everything out and to stamp out any remaining noise on the borders of the image. Binary thresholding was explained above under section 2.2. Finally, we get our output image below in Figure 2. The Edge Detection image was grabbed after the thresholding operation. The Mask image was gathered after the largest contour, morphology and binary thresholding operations. The final extracted image can be seen on the right hand side.

Please note that some parts of the motherboard were destroyed just like in the example image in Figure 1, this was necessary in order to get rid of the surroundings, a few capacitors, resistors etc. were masked out near the bottom. The lighting gradient on the bottom of the motherboard proved especially troublesome. I believe this could have been fixed with another pass through of adaptive thresholding with a much smaller blocksize.

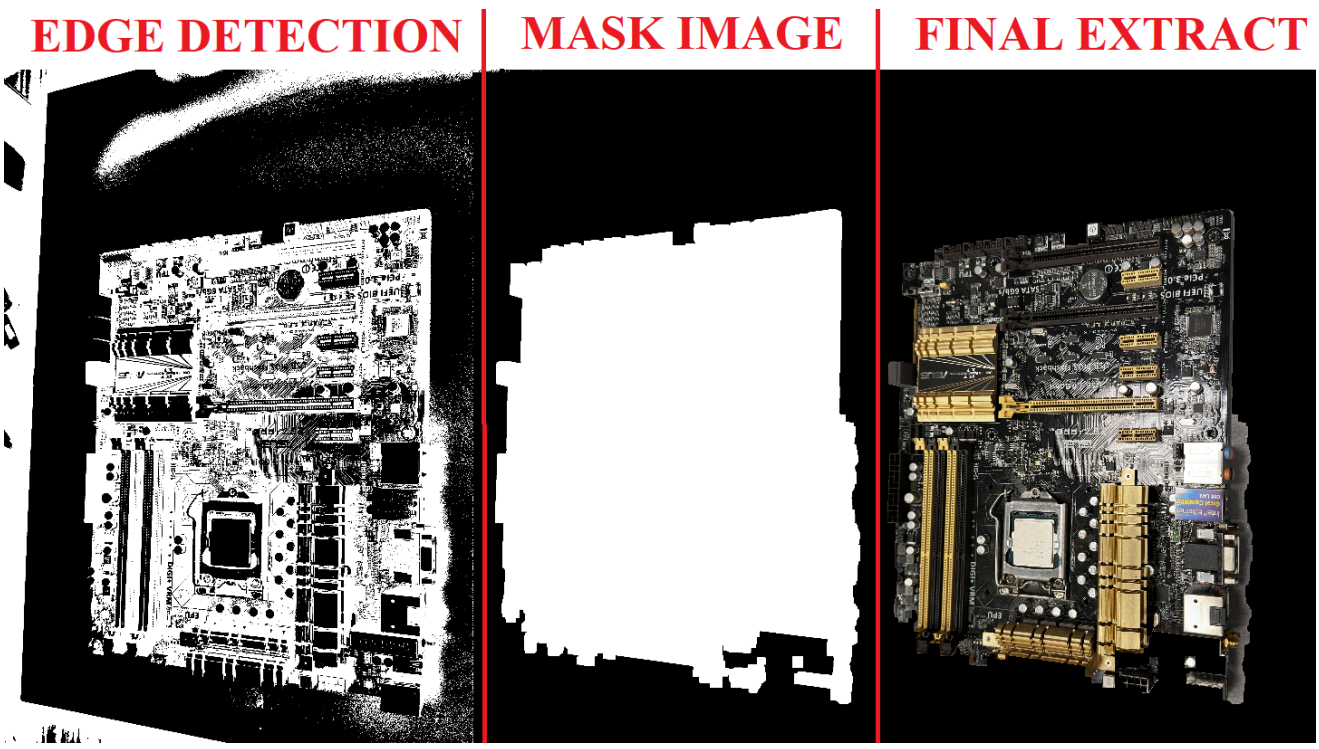

Figure 2: Edge Detection, Mask and Final Extracted Images

These images can be found in full size in the Github repository under /Motherboard

3.0 - YOLO Model Training, Evaluation & Iteration

As noted in the introduction of this report, 3 models were trained using YOLO nano version 11, see them again below:

- LOAV-PCB-v0
    - Epochs = 5, imgsz =900
- LOAV-PCB-v1
    - Epochs = 200, imgsz =900
- LOAV-PCB-v2
    - Epochs = 200, imgsz =1140

For all models trained, the batch size remained constant at 8, and thus the number of Epochs and the size of the images was all that was changed from iteration to iteration.

3.1 - LOAV-PCB-v0

Version 0 was an experimental model to gauge training times and to set a benchmark for how well the model was expected to perform with this task before devoting prolonged periods of time to training more complex models.

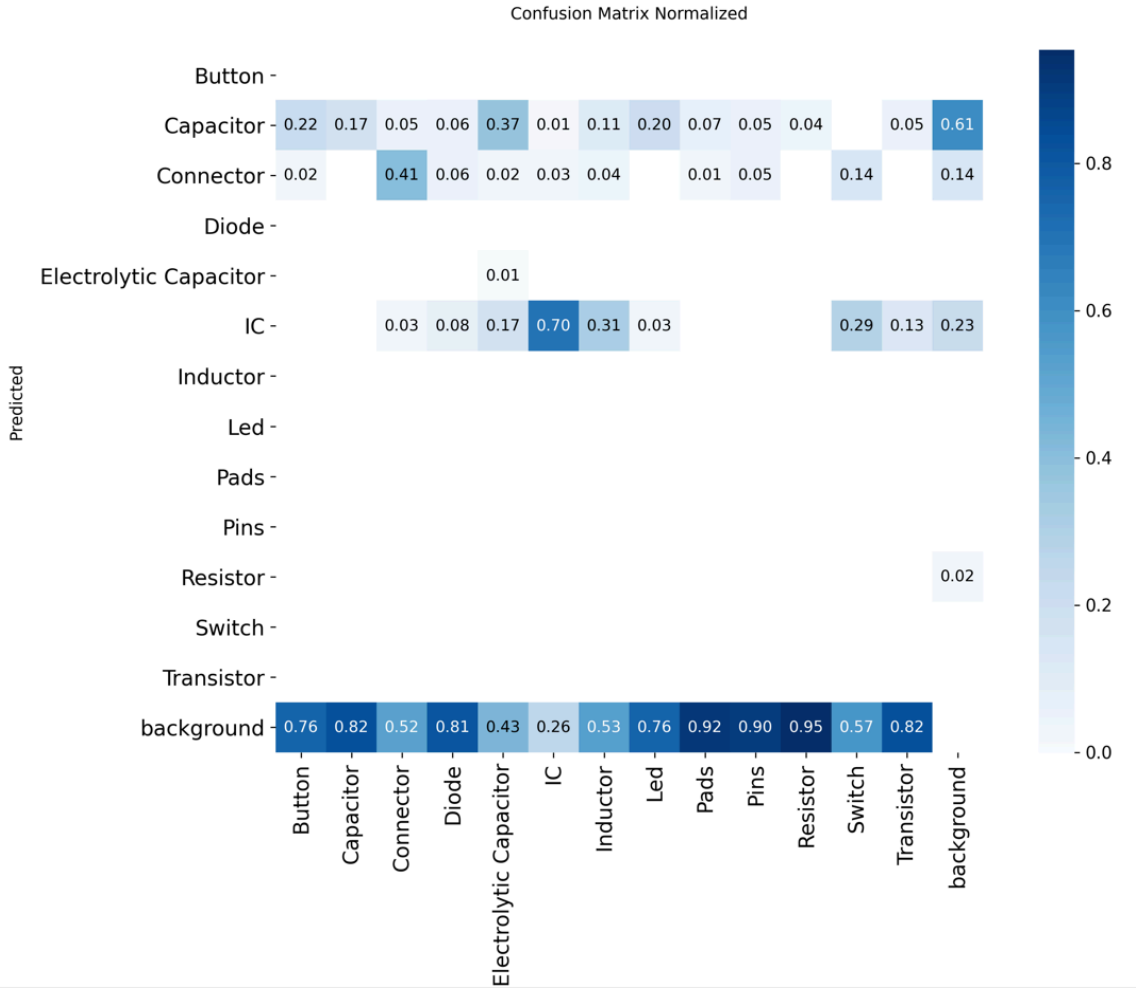

Figure 3: LOAV-PCB-v0 Confusion Matrix

Figure 3 above shows the confusion matrix of the 5 Epoch testing model, as expected, the model fails to learn anything of substance. It appears to predict background more than anything and seems to only be able to pick up bigger components such as capacitors, connectors and ICs. Smaller SMD components such as LEDs, resistors and transistors are never picked up by the model once. It makes sense that the bigger components which have the most distinct and easy to detect shapes would be learned first. More training required.

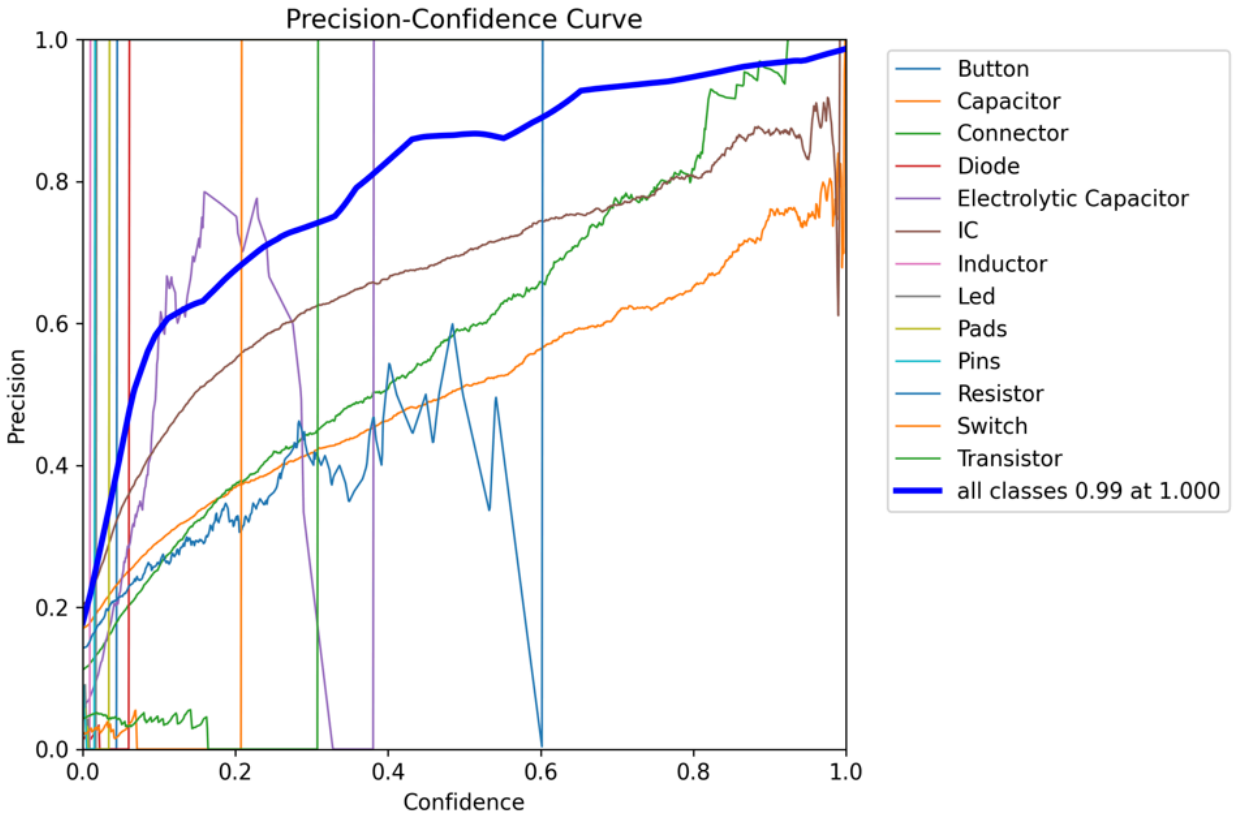

Figure 4: LOAV-PCB-v0 Precision-Confidence Curve

Figure 4 showcases the lack of learning quite well as most components hit 0 on the precision scale very far left in the graph. The majority of components hit 0 precision while there are a few that appear to have gained some traction, these are the larger components that were mentioned above under the confusion matrix figure 3.

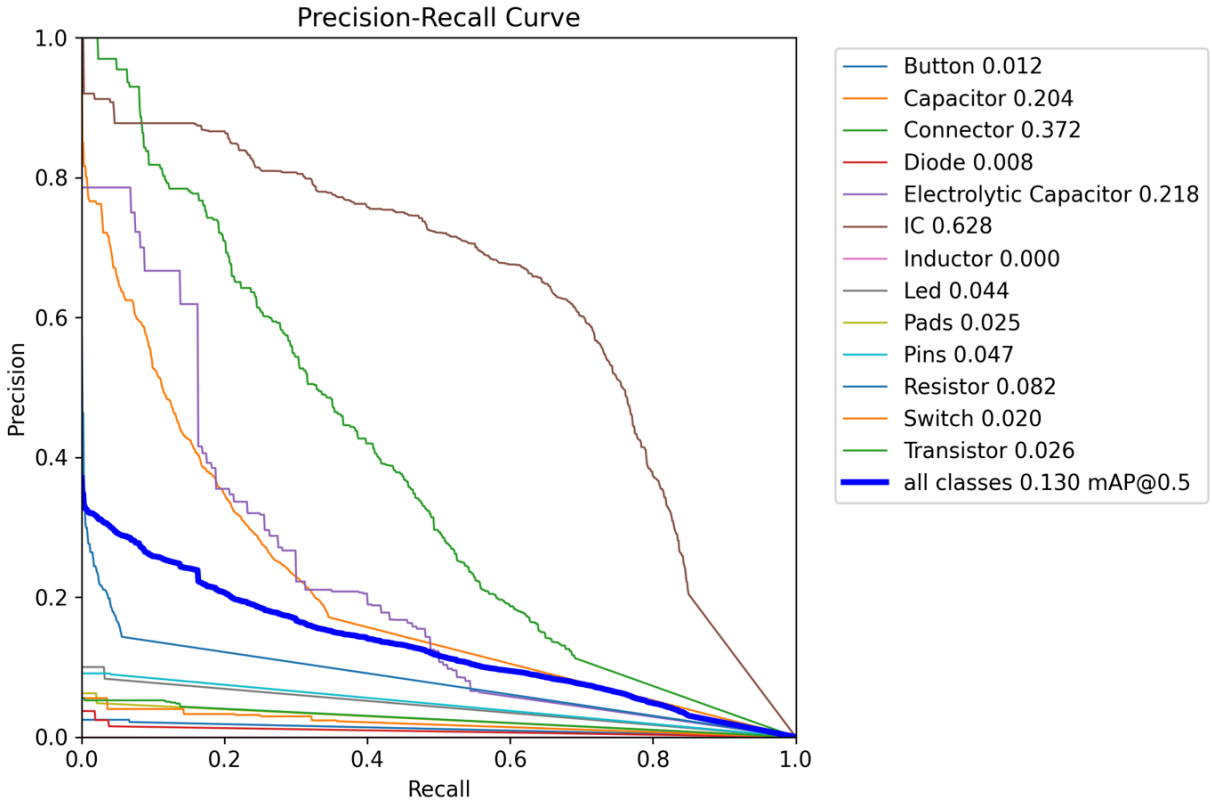

Figure 5: LOAV-PCB-v0 Precision-Recall Curve

Once again, we see very poor performance above in figure 5, if we take the top right corner to be perfect ideal performance, we see that the only part that gets remotely close is the IC. A good curve remains at a high precision even at a higher recall, but as we can see here, most components decrease at a fast rate, most of them not even starting at a high precision.

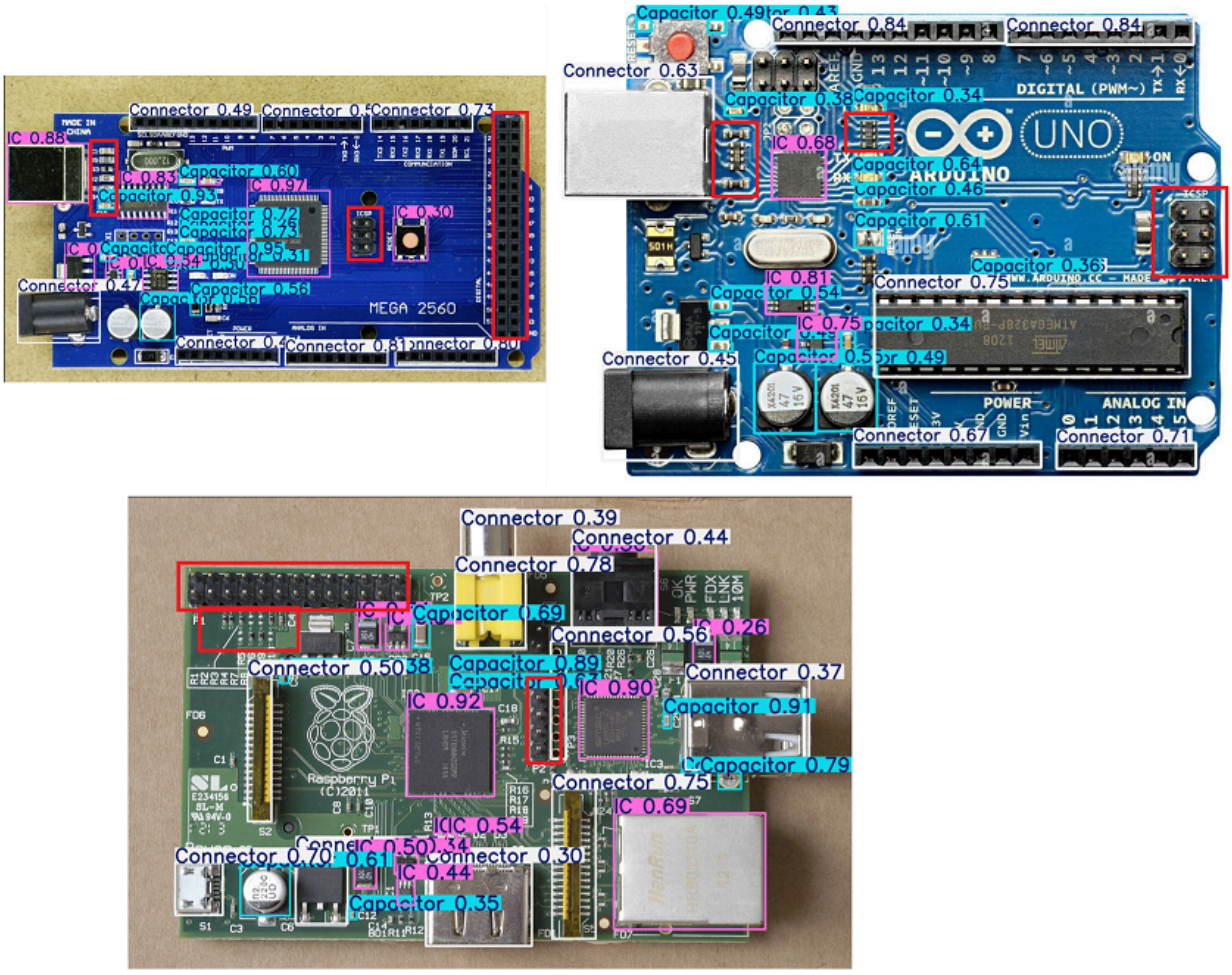

Figure 6: LOAV-PCB-v0 Evaluation Images

Figure 6 visualizes the results that we have come to understand through the graphs. We see that connectors, capacitors and ICs are detected but even then, some are missed. Some of these missed ones are highlighted in **RED.** We can see that the model has not detected a single resistor, pad or pin, nothing within that size range. Below in Figure 7, we see the same performance, once again, the model can only pick out a few connectors, capacitors and ICs. It is evident that more epochs will be required, and if even then, the model cannot pick up smaller components, it would make sense to increase the image size.

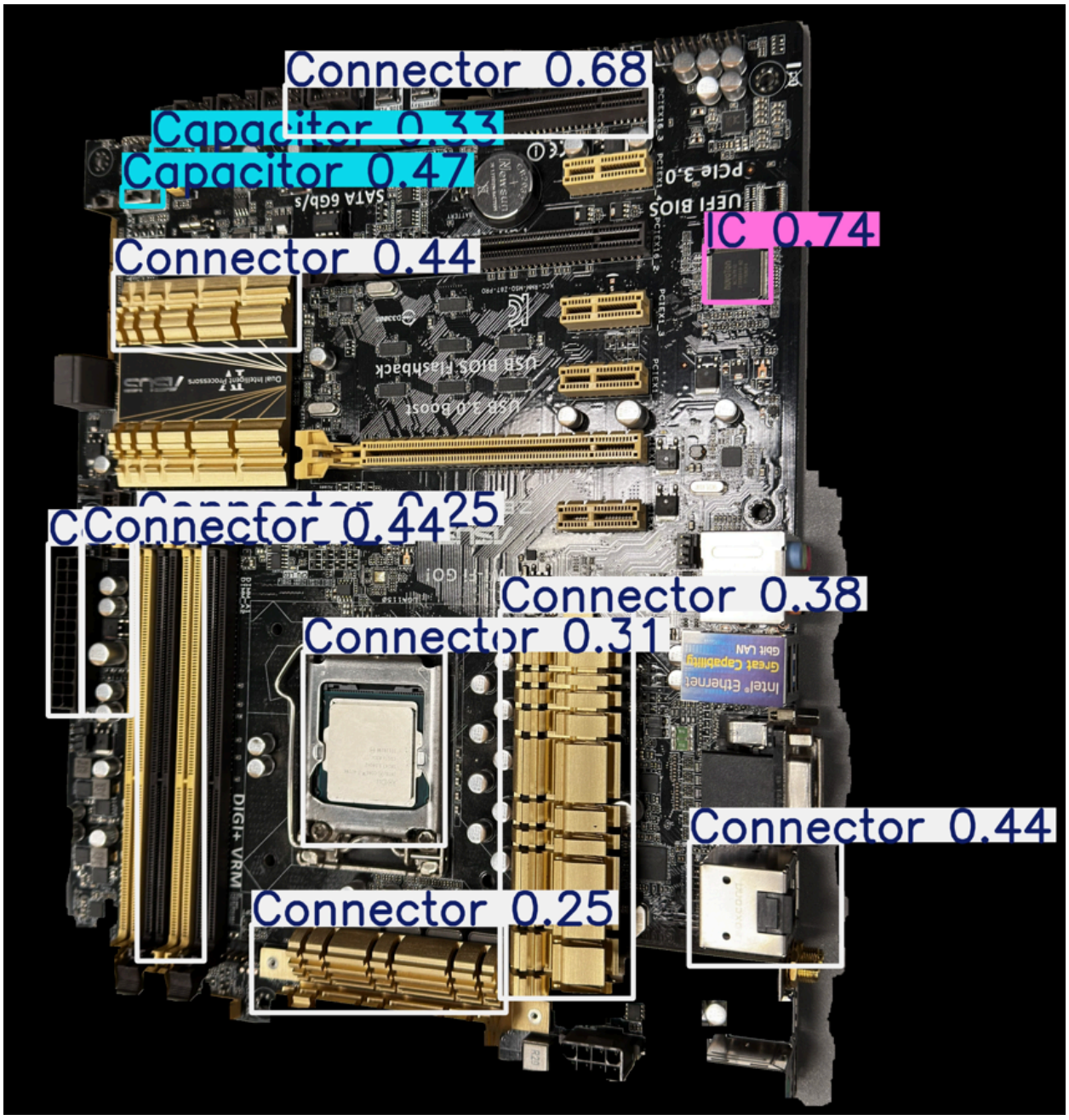

Figure 7: LOAV-PCB-v0 Motherboard Evaluation

3.2 - LOAV-PCB-v1

Version 1 kept all of the same parameters as version 0 but instead trained for 200 epochs, as we can see immediately below in Figure 8, the majority of the confusion matrix appears on the diagonal, meaning that the model is predicting accurately, with some hotspots around the rest of the matrix. We can see as well that the background being mixed up with components is still the most common occurrence across all components, with the majority of these erroneous predictions happening for smaller parts such as resistors, capacitors and pads. We also see some cross confusion between certain components, the biggest one being confusions between inductors and ICs as well as pads and resistors. Overall, many components have very high accuracy such as buttons, connectors, electrolytic capacitors, ICs and pins.

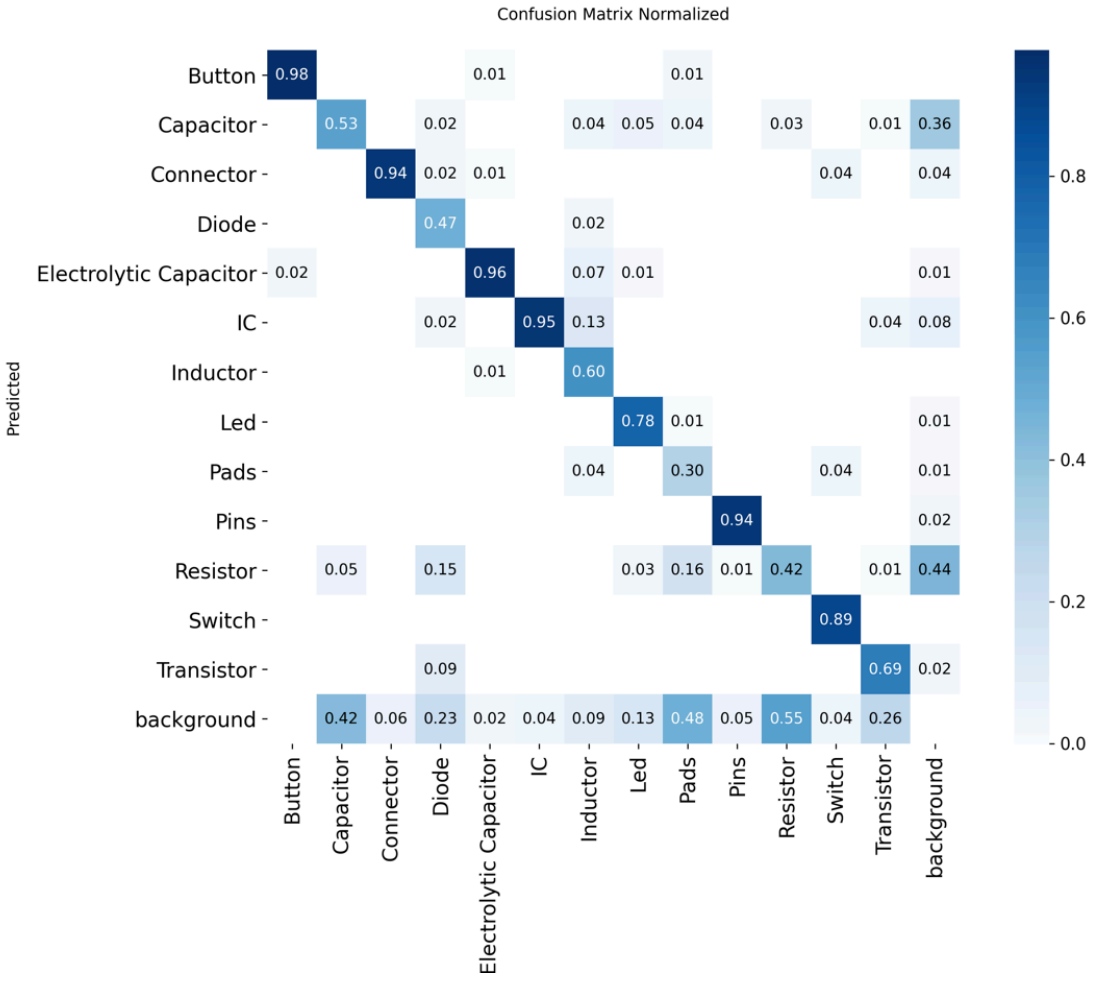

Figure 8: LOAV-PCB-v1 Confusion Matrix

Figure 9 below shows us that

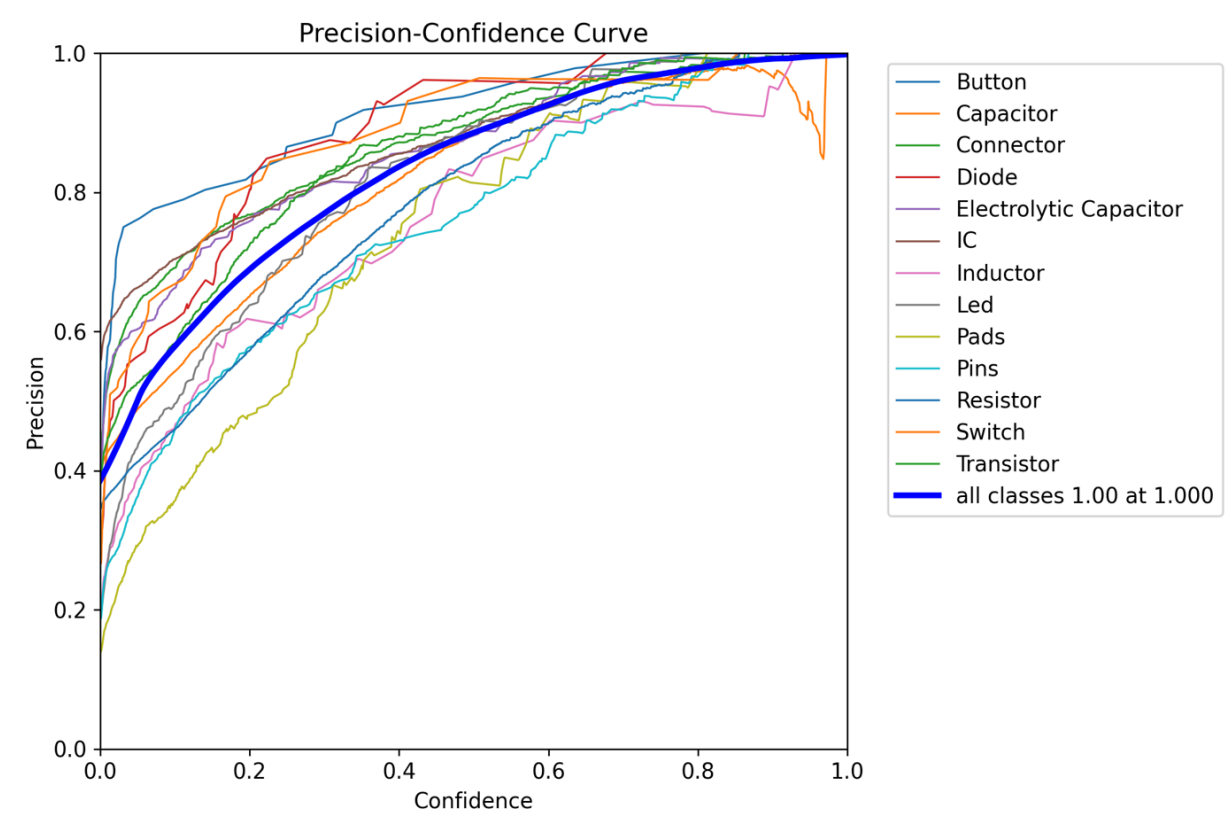

Figure 9: LOAV-PCB-v1 Precision-Confidence Curve

Unlike version 0, figure 9 above showcases a precision confidence curve more in line with good performance, all components converge towards the top right corner and approach 100% precision at a 100% confidence rate. It appears as though capacitors had a drop in precision near 95% confidence. This area marks a point in which the model was quite confident in the component benign a capacitor but it confused it with something else, and caused a drop in precision. This component it was confusing it with was likely just the background if we look at the confusion matrix. Overall, this is still very good performance and it appears that we should expect the model to miss some capacitors in the evaluation stage.

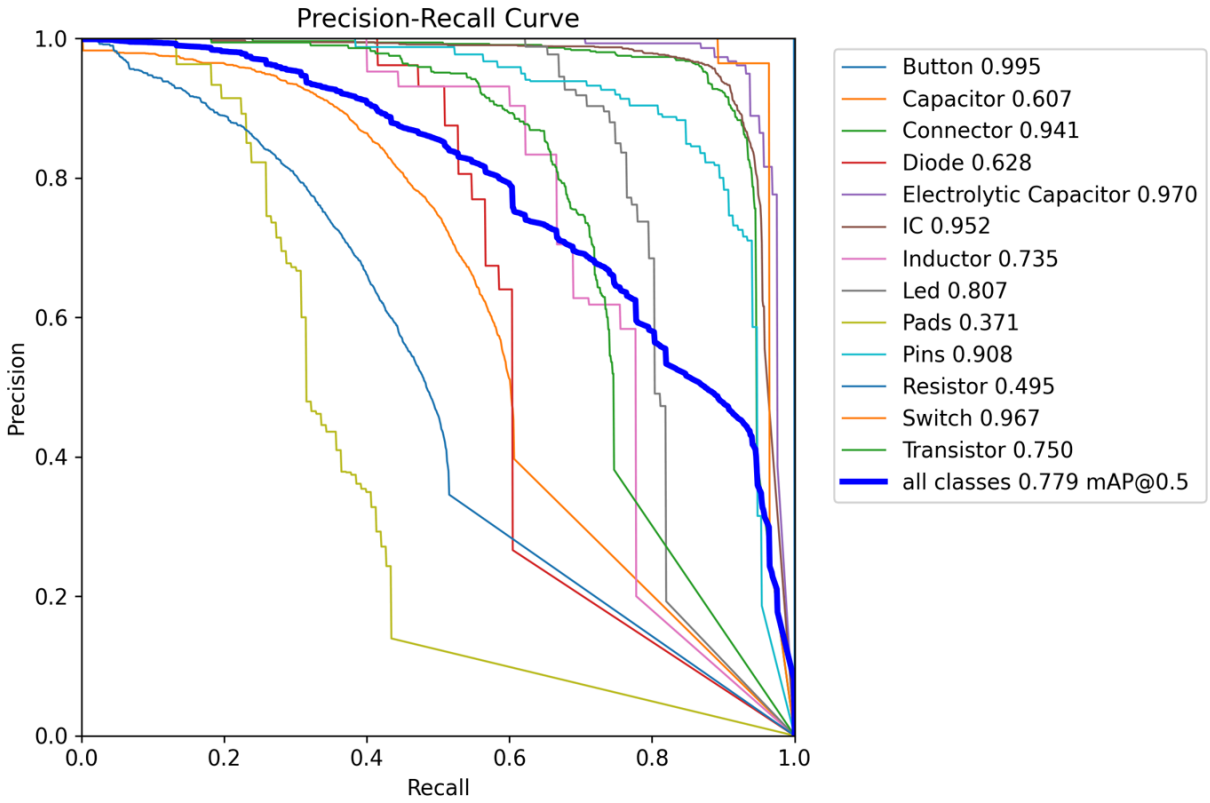

Figure 10: LOAV-PCB-v1 Precision-Recall Curve

Since a higher recall means that lower amounts of false positives are being detected, we can say that buttons, connectors, electrolytic capacitors, pins and switches are performing very well, we can see that these components remain at a high precision at all points of recall. On the other hand, components such as pads, resistors, diodes and capacitors perform quite poorly, this is understandable as these components are the smallest and in accordance with the confusion matrix, are most mistaken for the background.

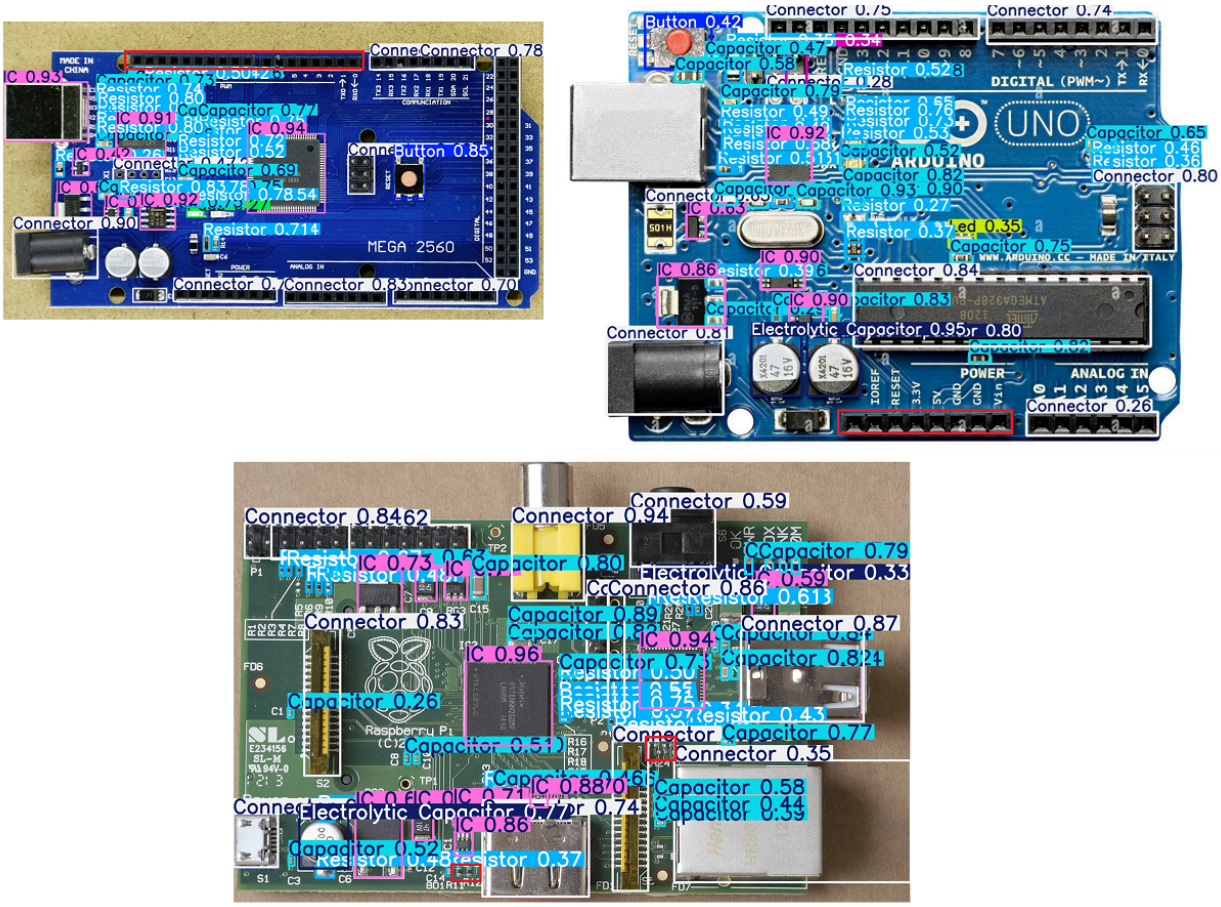

Figure 11: LOAV-PCB-v1 Evaluation Images

Figure 11 showcases this model's performance on the evaluation images; we can see here that many smaller components are recognized, such as capacitors and resistors. What is strange is some of the largest features like some of the connectors were not recognized despite the version 0 model being capable of doing so. These are highlighted in **RED.** However, it is certain that the performance of this model is much better than v0, finding many of the smaller components.

Figure 12: LOAV-PCB-v1 Motherboard Evaluation

Figure 12 above shows us that while this model did perform better than version 0, it was still not able to recognize the minute details of the motherboard. While credit should be given for its ability to recognize some capacitors, more connectors, and ICs, it still missed almost all of the electrolytic capacitors as well as many resistors and ICs. It is possible that this image was not masked properly and this is why it is experiencing so much trouble. It is also possible that the image data that trained the model consisted of purely top down images with minimal lighting effects, and this image is taken at an angle and may alter the models performance. It is clear that for the next mode, the image size should be increased as to better detect smaller components.

3.3 - LOAV-PCB-v2

Figure 13 shows us that model v2 with an increased image size has overall better performance across the diagonal, and has minimized the confusion of the background with components compared to model v1. Some components have improved marginally while others have not improved at all, but overall model v2 appears to be more accurate than model v1. Although, it appears the same 3 culprits of (capacitors, pads and resistors) are still the most commonly confused components.

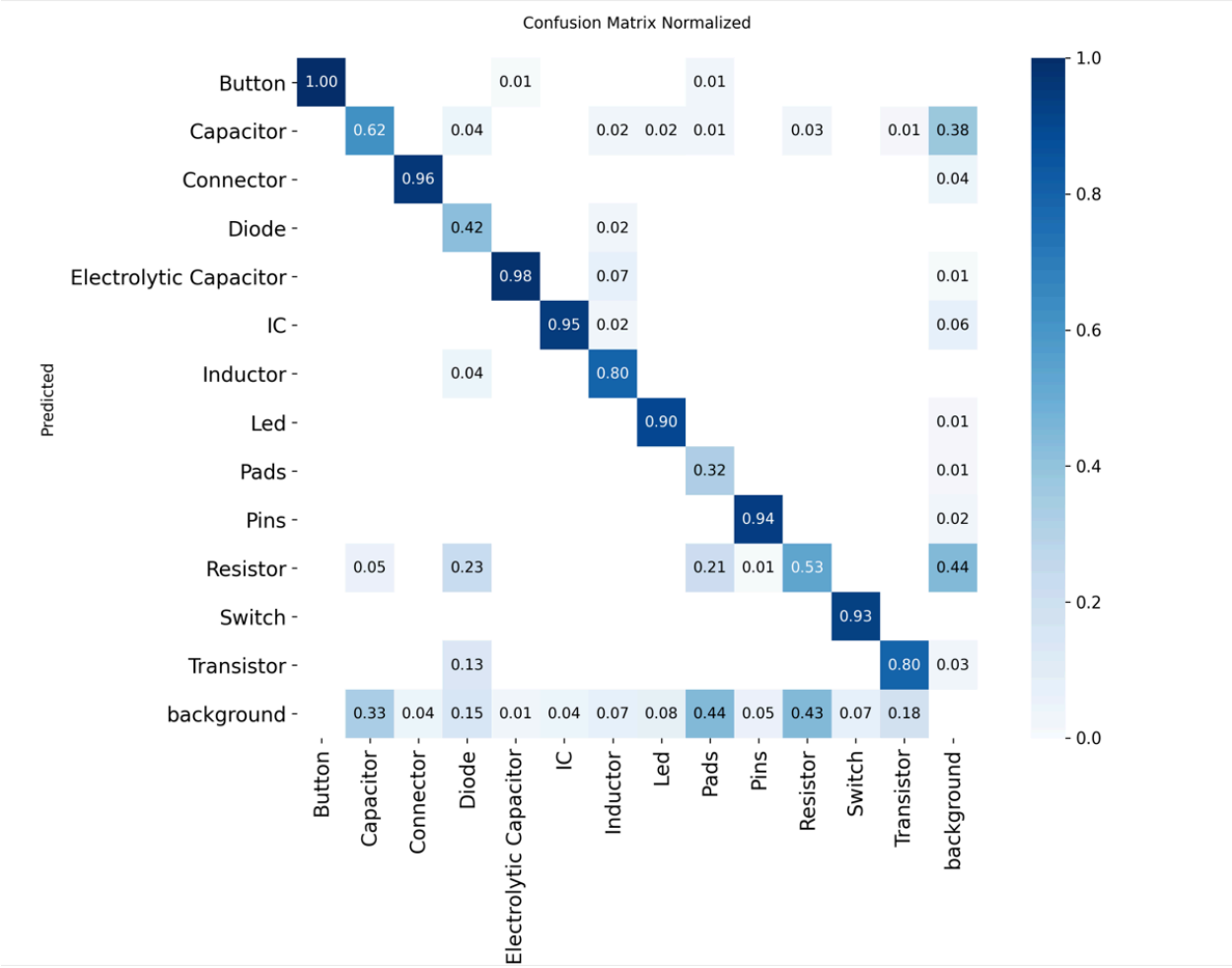

Figure 13: LOAV-PCB-v2 Confusion Matrix

Figure 14 showcases us that all classes reach a precision of 100% at an earlier confidence level of 0.996 compared to reaching 100% precision at 100% confidence of version 1. This is marginally better performance but still better. Overall, all of the curves appear to be following a tighter path compared to version 0, marking an overall increase in performance.

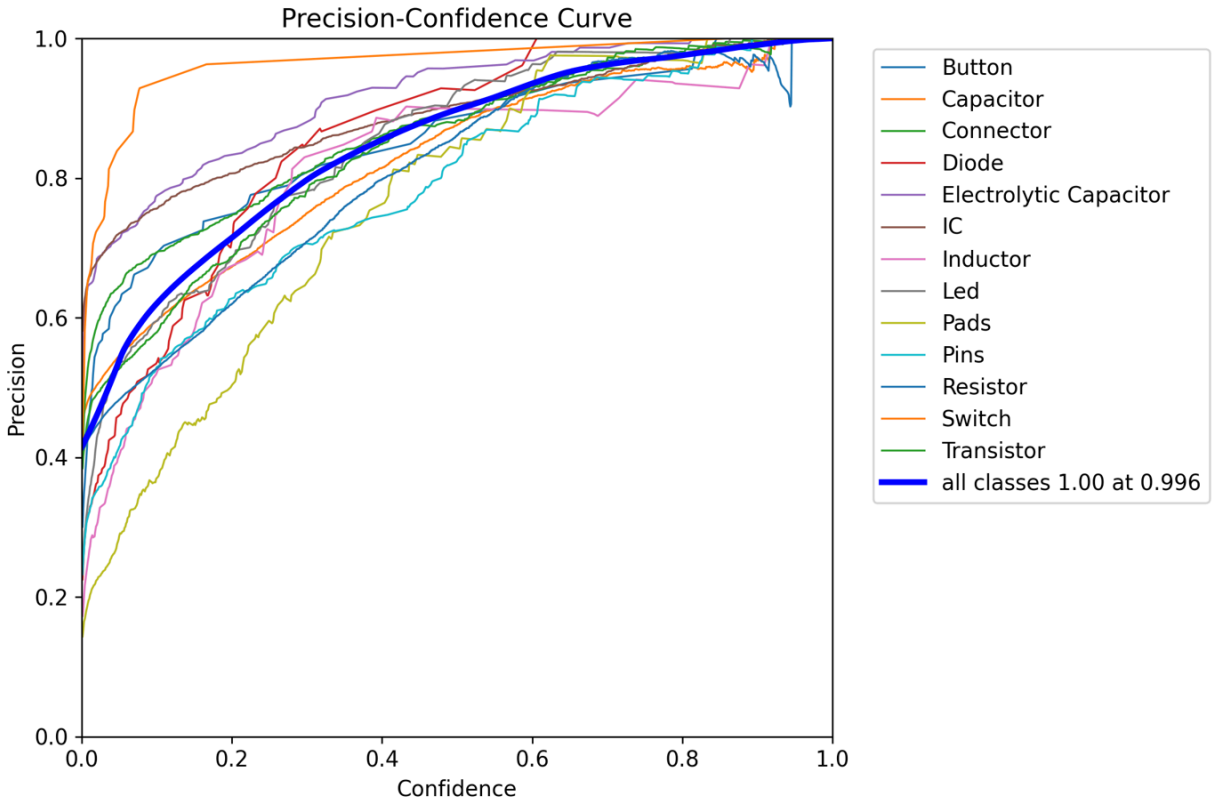

Figure 14: LOAV-PCB-v2 Precision-Confidence Curve

The mean overall curve here appears to have a more defined plateau which occurs higher on the precision scale compared to version 1, this marks overall performance as well. It should be noted that while we observe an overall increase in performance, certain components actually have a worse score compared to version 1, some of which are diodes and buttons, however these differences are extremely small. It appears as though the model still struggles the most with pads and diodes following close after. This is different from version 1 where resistors were 2nd worst in performance, here we can see slightly better performance from the resistors. It appears as though a higher image size had a positive effect on object detection of smaller components which makes sense in theory.

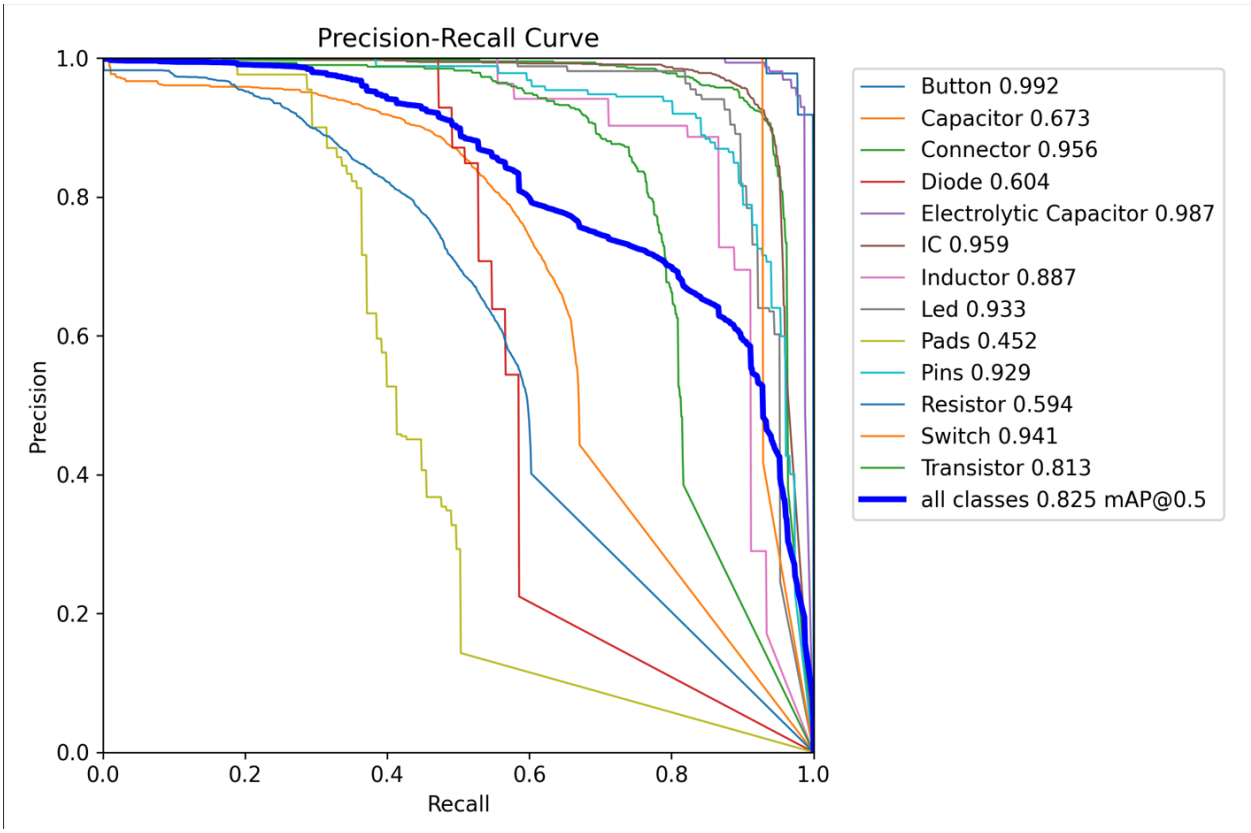

Figure 15: LOAV-PCB-v2 Precision-Recall Curve

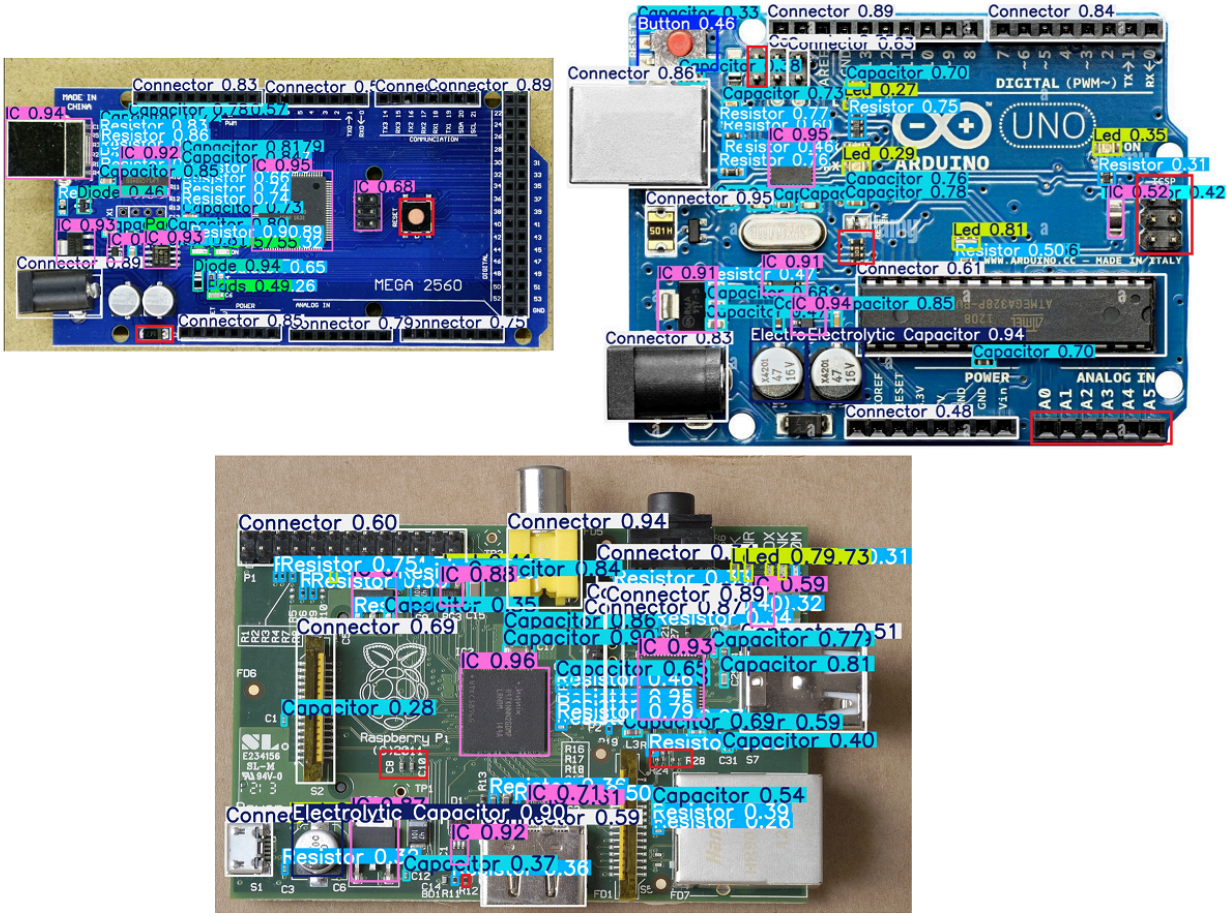

Figure 16: LOAV-PCB-v2 Evaluation Images

Once again, similar to model v1, some easy detections are missed such as the connectors highlighted in **RED**. However, we can see that just like model v1, model v2 was able to detect many smaller components compared to model v0. By inspection of figure 16, it appears as though model v1 and v2 are about equal in performance, but after looking at the metrics, it is evidence that model v2 is better than v1 but only by a small margin. Thus, its evaluations on both figure 16 above and figure 17 below look very similar to model v0. We have thus proved a positive correlation between image size and object detection, especially for smaller details such as smaller SMD components, and if my hardware permitted it, I would continue to increase the image size as I continued in this training and iterating process.

Figure 17: LOAV-PCB-v2 Motherboard Evaluation

4.0 - Conclusion

In conclusion, the LOAV-PCB project succeeded in creating an image masking pipeline to extract an image of a motherboard with minimal background noise, as well as train multiple object detection YOLOv11 nano models for PCB component recognition.

It was found that out of the 3 models trained, v1 and v2 were very similar (their only difference was in image size), and with an increase in image size, the v2 model was able to perform better on smaller SMD components such as resistors and capacitors, this difference was quite small however despite a mediocre increase in image size (900px vs 1140px). Despite it all, the confusion between the background and the components themselves was the highest source of confusion.

If I was given more time and resources for this project, I would return to it and possibly consider applying an image processing pipeline to all of the training data such that the confusion between the background and components could be minimized. This could be done using adaptive thresholding like it was done with the motherboard. Furthermore, I would increase the image size even further to allow the models to handle smaller components. Unfortunately, my current resources did not have enough VRAM to handle an image size much greater than 1140px. This concludes this report.