

|  |  |  |  |
| --- | --- | --- | --- |
| **Semester (Term, Year)** | Fall 2023 | | |
| **Course Code** | AER850 | | |
| **Course Section** | 1 | | |
| **Course Title** | Introduction to Machine Learning | | |
| **Course Instructor** | Dr. Faieghi | | |
| **Submission** | Project | | |
| **Submission No.** | 3 | | |
| **Submission Due Date** | Dec 17, 2023 | | |
| **Title** | Project 3 Report | | |
| **Submission Date** | Dec 18, 2023 | | |
| **Submission by (Name):** | | | **Student ID (XXXX1234)** | **Signature** | |
| Nima Dorali-Beni | | | Xxxx63711 | N.D. | |

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 /](https://www.torontomu.ca/senate/policies/)

Aerospace Assignment Cover as of May 2022

GitHub Link: <https://github.com/nimadb/AER_850_Project_2.git>

This project aimed to develop techniques for automatic detection of flaws in motherboards. Frist, the OpenCV python library was used for image processing to remove the background of the images containing the motherboards; and second, YOLOv8 was used to train a model to automatically detect flaws in the motherboards.

Image Preprocessing: The code started with preprocessing of the images. OpenCV contains lots of functions that are useful in preparing a data set to be fed to a training model. For this project, a single image was chosen: motherboard\_image.JPEG. In the code, the following functions were used to process the image: colour conversion to gray scale, morphology extraction, thresholding, contour detection, contour filtering, contour filling, another set of morphology extractions, dilation, another set of contour detection, and finally masking the binary image with the original-coloured image.

The image is first converted to grayscale since the functions required to extract the motherboard from the image detect the image features based on a pixel grayscale colour from 1-255. Then, a morphological extraction function was done to remove any noise in the image; this removed the noisy desk that the motherboard was sitting on. Next then, arguably the most important operation, thresholding was done to convert the grayscale to black and white based on the given thresholds. Various threshold types were used until this final one was chosen as it gave the best results (with respect to the ones that were used). Next, contour detection was done to find the first set of contours within the image. These were then filtered based on the lengths of the contours. The threshold values for the filtering were tuned for the image. Then the contours were filled; without this, the final mask would give a coloured image of lines. The next morphology functions were done to close the gaps between the contours to fill in a bigger shape. This was then dilated to capture any minute details that were missing. Lastly, another set of contour detection was done to find the full outer contour of the motherboard. These were all then converted to a mask to lay atop the original image. The result is shown below:

|  |  |
| --- | --- |
| A computer motherboard on a table  Description automatically generated  a) Original Image | A close-up of a computer motherboard  Description automatically generated  b) Gray Scale Image |
| A close-up of a computer motherboard  Description automatically generated  c) Noise Filtered Image | A close-up of a computer chip  Description automatically generated  d) Threshold Image |
| A computer motherboard with green lights  Description automatically generated  e) Contour Detection Image | A white square on a black background  Description automatically generated  f) Mask |
| A close-up of a computer motherboard  Description automatically generated  g) Final Extracted Image | |
| Figure 1: a) b) c) d) e) f) g) Images from Step 1 using OpenCV preprocessing. | |

The final extracted image shows still that some parts of the motherboard are still missing, mainly the bottom right USB-B jack, and that some parts of the table on which the motherboard is sitting on is still showing. A lot of attempts were made to correct for these issues. However, it was quite difficult to make it perfect as at some point it became a tuning problem. The Canny edge detection function gave many problems originally and was thus not used. Depending on the type of thresholding used, the canny function would try to pick up on all the minute edges in the table surface. This would sometimes even cause the program to crash. The multiple layers of contouring were found to be necessary to fully extract the required edges of the image and to fill them in, in order to extract all relevant motherboard information.

Model Evaluation

The normalized confusion matrix shows that the model has a very true positive rate for detection buttons, connectors, capacitors, ICs, Pins, and Switches. The model has the most trouble with resistors as it tends to predict them as just background; basically failing to identify them.

A screenshot of a computer

Description automatically generated

Figure 2: Normalized confusion matrix.

The precision confidence curve in Figure 3 shows the performance of the model in terms of precision based on confidence. As seen, as the confidence in a guess increases, the precission of the identification of the model increases. There are some exceptions such switches as they have a drop in precision with higher confidence. Overall, the model has a precision of 99% at 100% confidence.

A graph of different colored lines

Description automatically generated

Figure 3: Precision-confidence curve.

Figure 4 below shows that the model, as predicted in Figure 3 by the normalized confusion matrix, has the best performance when it comes to identifying buttons, capacitors, connectors, ICs, pins, etc. All the plotted lines that have the greatest area under the precision recall curve performed the best. The All-Classes line shows the overall performance of the model for all of the identifiers.

A graph of different colored lines

Description automatically generated

Figure 4: Precision-recall curve.

Figure 5 below shows the automatic identification of the components of the MEGA board. As seen, the connectors on the top and bottom of the board are missed. Most resistors on the left side of the board are also missed. The two capacitors are missed at the bottom as well.

A blue circuit board with green and red text

Description automatically generated

Figure 5: Mega-board prediction.

Figure 6 below shows the automatic identification of the components of the Arduino board. As with the MEGA board, many of the smaller pieces that are less elevated on the board are also missed such as the resistors and transistors. This time however the capacitors are identified while the button is missed.

A close-up of a circuit board

Description automatically generated

Figure 6: Arduino-board prediction.

Figure 7 below shows the automatic identification of the components of the Raspberry-pi board. As with the previous boards, only the largest components have been identified with a lot of the other components being missed. There is a miss-identification of a connector in the bottom right even!

A close-up of a circuit board

Description automatically generated

Figure 7: Raspberry-pi-board prediction.

Overall, the trained model could definitely perform better. There are my components that are missed in the prediction steps. Training with higher epochs and batch sizes might improve the capabilities of the model.