AER850 Project 3

Section 011

Professor Reza Faieghi

Due Date 12/15/2024

Name: Nazem Eylji

Student#: 500957556

Contents

[1. Introduction 3](#_Toc185198947)

[2. Steps 2-3 3](#_Toc185198948)

[2.1. Step 1: Image Preprocessing 3](#_Toc185198949)

[2.2. Step 2: Model Training 4](#_Toc185198950)

[2.3. Step 3: Model Testing and Predictions 7](#_Toc185198951)

[3. Conclusion and Future Improvements 8](#_Toc185198952)

List Of Figures

[Figure 1 - Final Extracted Image 5](#_Toc185198962)

[Figure 2 - Edge Detection Image 5](#_Toc185198963)

[Figure 6 - Precision Recall Curve 6](#_Toc185198964)

[Figure 5 - Precision Confidence Curve 6](#_Toc185198965)

[Figure 3 - Confusion Matrix Normalized 7](#_Toc185198966)

[Figure 7 - Arduno 7](#_Toc185198967)

[Figure 4 - RaspPi 7](#_Toc185198968)

[Figure 8 - ArdMega 8](#_Toc185198969)

[NazemEylji/Project-3: AER850 Project 3](https://github.com/NazemEylji/Project-3/)

# Introduction

This project focused on detecting components on a PCB (Printed Circuit Board) using advanced image processing techniques and a machine learning model. The overall workflow was divided into three steps: preprocessing the motherboard image to isolate the relevant region, training a YOLOv8-based model to detect PCB components, and testing the model on new unseen images. Each phase of the project was designed to ensure accurate detection while minimizing noise and unnecessary artifacts in the outputs.

# Steps 2-3

## Step 1: Image Preprocessing

The primary goal of the preprocessing step was to isolate the motherboard from its background and remove unnecessary details, such as shadows or irrelevant regions, as shown in Figure 2. Using OpenCV, the original motherboard image was first converted into a grayscale format to simplify further processing by reducing color information. This grayscale image enabled a precise binary thresholding operation, where a threshold value of 80 was applied to highlight the PCB and suppress the background. The result, demonstrated in Figure 2, shows how the edge detection process separated the PCB from its surroundings, though some edge artifacts and shadows persist. The selected threshold value was carefully tuned to balance sensitivity to components while minimizing excessive noise, laying the foundation for a clean and isolated extraction in subsequent steps.

To further refine the binary image, morphological operations were performed using a rectangular kernel of size 15x15. This helped close small gaps in the mask, ensuring the PCB area was fully enclosed. Contours were then extracted from the binary mask, and the largest contour was identified as the PCB region. A tight mask was drawn around this contour and applied to the original image to isolate the motherboard effectively, as shown in Figure 2. This figure illustrates the edge detection process and highlights the challenges encountered, such as residual shadows and edge artifacts that persist despite the morphological refinements. These imperfections, while minimized, required careful tuning of parameters to ensure that the main PCB region remained intact and clearly visible. The resulting processed image, demonstrated in Figure 1, showcases a clean extraction where the motherboard is effectively isolated with minimal background noise, providing a strong foundation for further model training. Finally, any area outside the PCB was replaced with black pixels, creating a clean output where only the motherboard remained visible. This processed image, saved as final\_extracted\_motherboard.JPEG, served as the input for training the detection model in the next step.

While the preprocessing successfully removed the majority of noise and irrelevant regions, the results still highlight some challenges, as demonstrated in Figure 2. The edge detection process, although effective in isolating the PCB region, shows that shadows and edge artifacts persist, especially in areas with uneven lighting. These artifacts required careful fine-tuning of the threshold value and morphological kernel, yet some residual noise remains visible in the mask. This can be seen in Figure 2, where certain edges of the PCB are not perfectly smooth or cleanly separated. Such imperfections may have introduced minor inaccuracies in the final extracted image (Figure 1), and addressing these issues would require further refinements in preprocessing to ensure consistent results under varying conditions.

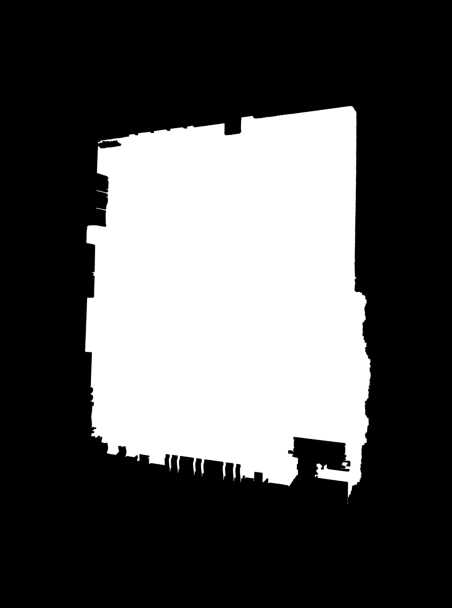
 

Figure 1 - Final Extracted Image

Figure 2 - Edge Detection Image

## Step 2: Model Training

For detecting components on the PCB, the YOLOv8 model was chosen due to its efficiency and accuracy in object detection tasks. Specifically, the YOLOv8 nano variant was used to ensure faster training while maintaining satisfactory performance. The training dataset was prepared and configured using a data.yaml file that defined the paths to training and validation images, as well as class labels for the various PCB components.

To enhance the model's ability to detect small and complex objects on the PCB, the image resolution was set to 1024x1024 pixels, ensuring finer details were captured. The training process spanned 120 epochs, which provided enough learning iterations to achieve strong performance without overfitting the model. A batch size of 8 was selected, balancing computational efficiency and learning stability. The AdamW optimizer was employed to improve weight regularization, and a learning rate of 0.001 ensured gradual and stable convergence. Additionally, a weight decay of 0.0005 was applied to prevent overfitting.

A graph of different colored lines

Description automatically generatedA graph of different colored lines

Description automatically generatedThe Intersection over Union (IoU) threshold was configured at 0.25, which allowed the model to refine bounding box predictions by suppressing overlaps and retaining the most relevant detections. Similarly, a confidence threshold of 0.25 was applied to filter out predictions with low certainty, resulting in cleaner and more meaningful outputs. The training logs, model weights, and predictions were saved under the directory runs/pcb\_component\_detection for later evaluation.

Figure 6 - Precision Recall Curve

Figure 5 - Precision Confidence Curve

Figure 6, the **Precision-Recall Curve**, demonstrates the performance of the trained model across various component classes. It highlights the model's ability to achieve high precision for components like "Buttons," "Connectors," and "Capacitors," while also revealing areas of improvement for smaller components like "Resistors" and "Switches," where precision drops slightly. Complementing this, Figure 5, the **Precision-Confidence Curve**, shows how precision increases as the confidence threshold rises, validating the decision to set a higher confidence threshold during predictions.

A screenshot of a graph

Description automatically generatedTo further analyze the model's detection performance, the **Confusion Matrix** in Figure 3 provides a normalized view of classification accuracy for each component class. The matrix highlights strong performance for commonly occurring classes like "Capacitors" and "Connectors" while also indicating slight misclassifications for smaller or overlapping components. This insight emphasizes the challenges associated with detecting fine-grained details and overlapping objects, particularly in densely populated regions of the PCB.

Figure 3 - Confusion Matrix Normalized

A close-up of a circuit board

Description automatically generated**A close-up of a circuit board

Description automatically generated**The predictions generated by the trained model were evaluated on unseen test images of different PCBs, including **RaspPi** (Figure 4), **Arduino** (Figure 7), and **ArdMega** (Figure 8). These figures visually demonstrate the bounding boxes drawn around the detected components, with confidence scores overlaid to indicate detection certainty. While the model successfully identified most components with high confidence, minor overlaps and missed components were observed in denser regions, particularly in Figure 8. This highlights the importance of further optimizing the model's IoU threshold and confidence settings to reduce false positives and improve component coverage.

Figure 7 - Arduno

Figure 4 - RaspPi

A blue circuit board with white text

Description automatically generatedOverall, the training process effectively leveraged YOLOv8’s strengths to achieve accurate detection across multiple PCB images. However, as illustrated in Figures 4, 7, and 8, future refinements, such as increasing resolution further or fine-tuning augmentations, could improve the model's performance, particularly for smaller and overlapping components.

Figure 8 - ArdMega

## Step 3: Model Testing and Predictions

The trained YOLOv8 model was tested on new unseen images to evaluate its performance and generate predictions. The test images were stored in a dedicated directory named test\_images, ensuring a clear separation from the training data. The model weights from the training phase were loaded, and predictions were performed with refined settings for confidence and IoU thresholds. This allowed the model to produce cleaner results by reducing low-confidence predictions and minimizing overlapping bounding boxes.

The predictions were saved in a separate directory named predictions\_refined under the runs folder. Each test image was processed, and the model output included bounding boxes and confidence scores for the detected components. The results were visualized and stored, providing a qualitative assessment of the model's performance.

Despite achieving satisfactory results, the model predictions were not perfect. In certain cases, overlapping components caused confusion, resulting in duplicate bounding boxes or incorrect classifications. Additionally, smaller components at the edges of the PCB were sometimes missed. This indicates that further tuning of the IoU threshold, confidence settings, or additional data augmentation could improve the model's accuracy.

# Conclusion and Future Improvements

This project successfully demonstrated a workflow for PCB component detection using a combination of image preprocessing and machine learning techniques. The preprocessing ensured that the model received clean, high-quality input images, while the training and testing phases showcased the ability of YOLOv8 to detect components effectively. The use of refined hyperparameters, including higher image resolution and tighter thresholds, contributed to the overall success of the project.

However, there are areas where improvements could be made. Shadows and lighting inconsistencies during preprocessing can still introduce noise into the images, which may affect the model's accuracy. Furthermore, the detection of small or overlapping components remains a challenge, highlighting the need for future refinements in training strategies and model settings. Increasing the resolution or experimenting with different object detection architectures may further enhance detection performance. Overall, this project provides a solid foundation for PCB component detection and offers several avenues for continued optimization and improvement.