AER850: Intro to Machine Learning

Project #3
Deadline: 23:59 December 17, 2023

---

1 Background & Introduction

In circuit manufacturing, ensuring accurate component placement traditionally relies on labor-intensive manual checks. This project seeks to automate and streamline this process by integrating computer vision systems. High-resolution images of printed circuit boards (PCBs) leaving the assembly line are processed using a machine learning model to automatically detect and verify component placement. This approach aims to enhance manufacturing efficiency by reducing reliance on manual inspection and improving overall accuracy in component placement.

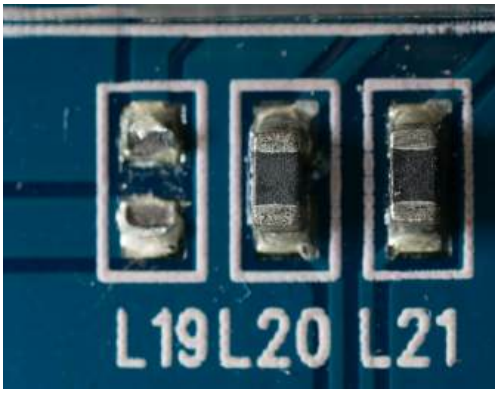

Figure 1: Missing Component Example

2 Project Outline

This project includes two major sections. Image processing and component detection. For the first section, you are to use **OpenCV** to mask out an image of a motherboard. The second section involves using **YOLOv8** to classify components from a PCB. The packages required for all steps below: **OpenCV, Ultralytics, PyTorch, numpy, and Pillow**.

To successfully execute these tasks, **OpenCV** will be harnessed in the initial phase for precise image masking, ensuring the extraction of relevant features from the motherboard image. Subsequently, **YOLOv8**, the latest YOLO model from Ultralytics, will be employed to conduct component classification on the PCB.

2.1 Step 1: Object Masking - 30 Marks

Object masking is a crucial tool in computer vision as it enables machines to precisely identify and isolate specific objects within images, fostering enhanced image understanding. OpenCV tools are essential for working with image processing. There are a few steps required to do this efficiently.

- Thresholding in image processing is a technique used to segment an image into regions based on pixel intensity. This can be done using the threshold function in OpenCV.
- Edge detection can be done with two methods:
  - Corner detectors are algorithms in computer vision that identify and locate points in an image where there is a significant change in intensity or color. Types of corner detectors include Canny or Harris.
  - Contour detection is a computer vision technique that involves identifying and tracing the boundaries of objects within an image, highlighting the regions where there are significant changes in intensity, color, or texture.
- The area of the contour or the size of the line can be used to filter out smaller lines or small contours.
- Once a contour is made and a mask is extracted, the *cv2.bitwise_and()* operator can be used to extract the PCB from the background.

The image you will work with is found on D2L as **motherboard_image.JPEG**. Your extracted image should be similar to the following example:

Figure 2: Extracted Motherboard Image

2.2 Step 2: YOLOv8 Training - 30 Marks

For this stage you are to use Ultralytics to train a model to detect components on a PCB. It is recommended for you to use YOLOv8 nano as your pretrained model. From this model, fine-tuning can be done to cater the model to our use case. More information about this model can be found at: YOLOv8 Documentation. Since this model is pretrained, no modifications to the structure of the neural network are required. However, due to the size of the dataset used for this training, the time for training will be much longer than that of previous projects.

- Using the dataset provided, train a YOLOv8 model using Ultralytics' *model.train()* tool. Three key hyperparameters you may want to include are:
  - **epochs** - defines the amount of iterations to be run. With regards to epochs, remain below 200.
  - **batch** - defines the number of images to be analyzed in each epoch
  - **imgsz** - defines the image size to be used (takes a single value, e.g. 640, 800, 1200). Minimum recommended size is **900**, as small components must be recognized.
  - name - may also be included to define the name of the model to be trained

Training with the dataset provided will take a long time if your computer does not have a graphics card capable of accelerating machine learning tasks. For this project, we **highly recommend using Google Colab** for this section of the project. Training times may go above 1 hour based on the hyper parameters used. Upload the data to your Google Drive, this can be accessed from Google Colab (mount Google Drive). Make sure when using Google Colab, the current selected runtime is set to **Quadro T4/T4 GPU**. This will provide GPU acceleration for training your model. Google Colab may have some usage limits, 12 hours of GPU usage can cause your account to be on hold. If you run into this error try a different account or feel free to send an email for assistance.

2.3 Step 3: YOLOv8 Evaluation - 10 Marks

Once your model is trained, you must evaluate three images found in the **Evaluation** folder of the dataset. You will need to use the *model.predict()* function from Ultralytics to test each image. Some components may not be detected or may be mislabeled based on how many epochs were run. In terms of accuracy your evaluations should fall between the following two runs:

Figure 3: Low Epoch Evaluation Results

Figure 4: High Epoch Evaluation Results

The full resolution images can be found on D2L.

3 Submission - 30 Marks

For the submission of this project, a formal report is required. Code submission will be through GitHub. Create a new repository for this project and submit your report with the link to your GitHub repository through D2L. For each subsection the following is required:

3.1 Object Masking

A written explanation of what methods were used to detect the edges of the PCB and extract it from the background is required. If the masking was not exact, explain why it was not, and how it could be improved. Include figures for your edge detection, mask image, along with the final extracted image.

3.2 YOLOv8 Training & Evaluation

Explain how you trained your model and what could be done to improve it's detection capabilities. Include the figures and a brief explaination of the normalized confusion matrix, precision-confidence curve, and the precision-recall curve from your training results. These can be found in the folder for your model generated in *runs>detect>model name*. Include the three test images along with a brief explaination of what was missed or confused by the model.