MAIS 202 - Project Proposal: Automated PCB Defect Detection

Mia Valderrama-Lopez

October 2024

1. Choice of Dataset

I have chosen the **PCB Defect Dataset** from Kaggle: `https://www.kaggle.com/datasets/norbertelter/pcb-defect-dataset`. This dataset contains more than a thousand images of Printed Circuit Boards (PCBs) across six common defect types: Missing Hole, Mouse Bite, Open Circuit, Short, Spur, and Spurious Copper plus a "good" class for functional boards. As an electrical member of the Robotics team working on an Automated Underwater Vehicle (AUV), ensuring the integrity of our PCBs is really important. Plus, I have been doing a lot of embedded code lately and would like to gain more knowledge concerning PCBs.

2. Methodology

a. Data Preprocessing

The dataset is clean and ready to use for image classification. Basically, I need to teach the model to spot the visual differences between a normal PCB and one with defects. To get the data ready, I'll:

- **Resizing:** Standardize all images to 224×224 pixels for consistent input dimensions
- **Normalization:** Scale pixel values to [0, 1] range to stabilize training
- **Data Augmentation:** Apply random transformations such as rotations and flips to improve model generalization
- **Train-Test Split:** Use an 80-20 split with stratified sampling to maintain class distribution

b. Machine Learning Model

The goal is **multi-class image classification** to identify specific PCB defect types among 7 classes (6 defects + "good PCB").

Primary Model: Convolutional Neural Network (CNN) - I'll build one from scratch. CNNs are perfect for images because they automatically learn to recognize patterns like edges, shapes, and textures, kind of like how our brain processes visual information layer by layer.

Alternative Model: Transfer Learning using pre-trained models like ResNet50 or MobileNetV2. This is like getting a head start - instead of training from scratch, I can take a model that's already really good at recognizing images (since it learned from millions of photos) and just tweak it to specialize in spotting PCB defects.

Pros and Cons:

- **Building my own CNN:** I'll learn more about how neural networks work and can design it exactly how I want, but it might need more data and time to get good results.
- **Using Transfer Learning:** I can get better accuracy much faster since the model already knows how to see, but the model is bigger and it's harder to understand why it makes certain decisions.

c. Evaluation Metric

To see if my model is actually working, I'll check:

- **Confusion Matrix** - a table that shows what the model gets right and wrong
- **Accuracy** - overall how often it's correct
- **Precision & Recall** - precision measures how many of the defects it finds are real, recall measures how many actual defects it catches
- **F1-Score** - balances both precision and recall (the most important metric!)

If I just guessed randomly among the 7 types of defects, I'd only get about 14% accuracy. I'm aiming for **at least 95% accuracy** and a **F1-score above 0.90**, which would mean the model is actually useful for finding real defects.

3. Application

The final model will be integrated into a simple web application using Streamlit or Flask:

- **User Input:** A drag-and-drop interface or file selector for uploading PCB images (png, jpeg).
- **Output:** The input image displayed with defective areas highlighted, along with the predicted defect class and model confidence score or simply will display a defective or good output.
- **Use Case:** Provides immediate, actionable feedback for quality control technicians in manufacturing environments !