CircuitGuard: Automated PCB Defect Detection System

Final Project Report

Submitted by:
Badal Dadwani
Infosys Springboard Virtual Internship 6.0
Batch 5

CircuitGuard: Automated PCB Defect Detection System

1. Introduction

Printed Circuit Boards (PCBs) are fundamental to modern electronic systems, providing mechanical support and electrical routing for components and integrated circuits. Even small defects—such as missing holes, shorts, spurious copper artifacts, or broken traces—can compromise performance, reduce reliability, or render an entire device non-functional. As electronic devices continue to shrink and incorporate increasingly dense circuitry, inspection requirements have become more demanding.

Traditional inspection methods such as manual visual inspection or Automated Optical Inspection (AOI) frequently suffer from limitations. Manual inspection introduces human error and inconsistency, while AOI systems are expensive, often inflexible, and depend heavily on handcrafted rules that do not generalize well across different PCB layouts or defect types.

This project addresses these challenges by developing **CircuitGuard**, a complete software-based inspection pipeline that uses image processing and deep learning to automatically detect and classify PCB defects. CircuitGuard compares a test PCB with a golden reference, isolates defect regions, classifies them into standard categories, and produces a comprehensive annotated report. The system is designed to be flexible, scalable, and cost-effective, making it a viable alternative to traditional hardware-based inspection systems.

2. Problem Description

Modern manufacturing environments frequently handle multiple PCB designs, each requiring precise, consistent, and automated inspection. Variability in illumination, image capture conditions, and PCB alignment makes rule-based inspection difficult, while manual inspection is not scalable.

The primary objectives of this system are:

- To automatically determine the correct golden PCB corresponding to an uploaded test PCB.
- To detect structural deviations between the test image and the golden reference.
- To extract and isolate regions potentially containing defects.
- To classify each identified region into one of six standard PCB defect categories.
- To generate visual and machine-readable reports summarizing all detected defects.

By combining classical computer vision and deep learning, CircuitGuard provides a robust, efficient, and automated quality-inspection framework.

3. Dataset Description

The dataset used originates from the Kaggle PCB Defects dataset. It contains 1386 images of PCBs exhibiting six defect classes: missing hole, mouse bite, open circuit, short, spur, and spurious copper. Each class is represented by hundreds of labeled examples. The dataset provided defect-level crops, enabling effective supervised training.

A custom script was used to split images into training and testing partitions while maintaining class balance.

4. Preprocessing and Image Preparation

Several preprocessing steps were applied to ensure uniformity and model compatibility:

• Conversion to RGB when needed.

• Resizing all images to 128×128 pixels.

• Normalization to stabilize neural network training.

• Conversion to tensors for PyTorch.

For defect detection, preprocessing included grayscale conversion, absolute difference computation, binary thresholding, and contour extraction to identify potential defect regions.

5. Image Subtraction Based Defect Detection

The core principle behind defect localization is **image subtraction**.

1. Both PCB images are converted to grayscale.
2. `absdiff()` computes the pixel-wise absolute difference.
3. Thresholding converts the difference map into a binary mask.
4. Contours on the thresholded mask represent regions where structural differences exist.
5. Bounding boxes are derived from contours to extract Regions of Interest (ROIs).
6. Padding is added to ensure complete coverage of each potential defect.

This approach reliably highlights anomalies while ignoring background variations and noise.

6. Deep Learning Model: EfficientNet-B4

A classification model based on EfficientNet-B4 was trained on the defect dataset. EfficientNet-B4 balances accuracy and efficiency, making it suitable for real-time inspection.

Training configuration:

- Optimizer: Adam
- Loss function: Cross-entropy
- Batch size: 32
- Epochs: 20

The model outputs one of six defect types along with a confidence score. The final test accuracy reached 99 percent.

Classification Report Summary:

- Precision: 0.99
- Recall: 0.99
- F1-Score: 0.99
- Total test samples: 2080

7. Automatic Golden PCB Identification

In production environments, multiple PCB designs may exist. To eliminate manual golden selection, the system includes an identification module that determines the correct golden

PCB based on the uploaded test image.

This uses ORB (Oriented FAST and Rotated BRIEF) feature detection:

• Keypoints and descriptors are extracted from the test PCB.

• Each golden PCB undergoes the same extraction.

• A brute force matcher counts descriptor matches between the test PCB and each golden PCB.

• The golden board with the highest match score is selected.

The match score represents the number of valid keypoint correspondences. The highest score indicates the most similar PCB layout.

8. ROI Classification and Annotation

For each valid contour-derived region:

1. The ROI is cropped from the test image.
2. The trained EfficientNet-B4 model classifies the ROI.
3. Confidence values are computed using softmax probabilities.
4. Bounding boxes and labels are drawn on the final annotated image.

Each detection entry includes:

- Defect category
- Bounding box coordinates
- Center point
- Confidence percentage

All detections are sorted by confidence for clarity.

9. Backend System Architecture

The backend system is built using Flask. Its responsibilities include:

- Loading and managing the classification model
- Handling automatic golden PCB identification

- Executing preprocessing, detection, and classification
- Producing annotated images
- Generating JSON and CSV reports
- Providing a stable API endpoint (`/detect`) for the frontend

The system is designed to be efficient and stateless, allowing easy scaling or deployment to production environments.

10. Frontend Interface

The frontend interface was designed using HTML, CSS, and vanilla JavaScript.
It enables users to:

- Upload a test PCB and optionally a golden PCB
- Enable automatic golden reference detection
- View annotated results directly in the browser
- Examine defect tables with bounding box and confidence information
- Download the annotated image, CSV summary, and JSON log

The layout is user-friendly and suitable for both academic demonstration and industrial inspection dashboards.

11. End-to-End Pipeline Workflow

The complete CircuitGuard process is summarized below:

1. User uploads a test PCB image.
2. If enabled, the system automatically selects the correct golden PCB.
3. The golden and test images are aligned through subtraction.
4. Thresholding and contour detection isolate potential defects.
5. EfficientNet-B4 classifies each ROI into one of six defect categories.
6. The final output includes an annotated PCB image and downloadable defect reports.

This workflow provides a fully automated inspection pipeline requiring no manual intervention.

12. Conclusion

CircuitGuard demonstrates that AI-driven software solutions can serve as highly effective and low-cost alternatives to traditional AOI systems.

With a combination of image subtraction, contour extraction, and deep learning, the system delivers:

- 98% defect classification accuracy
- Automatic golden PCB identification
- A complete, user-ready web interface
- Detailed JSON and CSV reporting

The system provides strong potential for real-world deployment in electronics manufacturing.

Future enhancements may include:

- Real-time video-based inspection
- More robust alignment and registration techniques
- Semantic segmentation for precise defect boundary detection
- Integration with factory automation systems