INFOSYS VIRTUAL INTERSHIP

CircuitGuard

Automated PCB Defect Detection & Classification System

Prepared By: ANUMULA DINESH REDDY

Student ID: HU22CSEN0101878

Email ID: danumula@gitam.in

Course: INFOSYS VIRTUAL INTERSHIP

Instructor: PRANAYA

Submission Date: 6th December, 2025

Abstract

Printed Circuit Boards (PCBs) form the backbone of all modern electronic systems. Manual inspection or traditional AOI (Automated Optical Inspection) systems often struggle with complex, fine-grained defects, are prone to human fatigue, and require time-consuming visual verification. The CircuitGuard project addresses these limitations by building a fully automated defect detection pipeline, integrating:

- Reference-based Image Subtraction,
- Contour-based Defect Region Extraction, and
- Deep Learning Classification (EfficientNet-B4)

The backend is powered by Python, OpenCV, and PyTorch, while the deployed application uses Streamlit for a modern UI. The trained classification model achieves ≥97% accuracy, meeting the project's primary performance benchmark. The system outputs:

- an annotated PCB board with bounding boxes,
- per-defect confidence metrics,
- and exportable CSV logs for documentation and analysis.

Chapter 1: Introduction and Problem Definition

1.1 Background & Need

PCB inspection ensures electronic reliability. Traditional inspection methods include:

1. **Manual Visual Inspection**
   Slow, subjective, error-prone, suffers from inspector fatigue.

2. **Basic AOI Systems**
   Good at rule-based checks but cannot perform robust classification.

3. **Deep Learning Approaches**
   Provide high repeatability and accuracy but require strong datasets and segmentation preprocessing.

CircuitGuard bridges these areas by implementing a hybrid approach:
Image subtraction → defect extraction → defect classification.

1.2 Problem Statement

The goal was to build a **complete PCB defect detection system** that:

1. **Automatically aligns and compares** a test PCB image with a defect-free reference template.

2. **Localizes defects** using image subtraction and morphological refinement.

3. **Extracts ROI patches** for each defect.

4. **Classifies the ROI** into the correct defect category.

5. **Generates annotated outputs** and downloadable logs.

6. **Powers a real-time web UI** for user interaction.

1.3 Final Deliverables

The project successfully produced:

- ✔️ Complete Preprocessing Pipeline (Alignment → Subtraction → Masking)
- ✔️ Connected-components based ROI extraction
- ✔️ Trained EfficientNet-B4 classifier (≥97% accuracy)
- ✔️ Full Streamlit Web Application
- ✔️ Backend integration with logs, predictions, and annotated image export
- ✔️ High-performance processing workflow (≤5 seconds per PCB image)

Chapter 2: System Architecture & Workflow

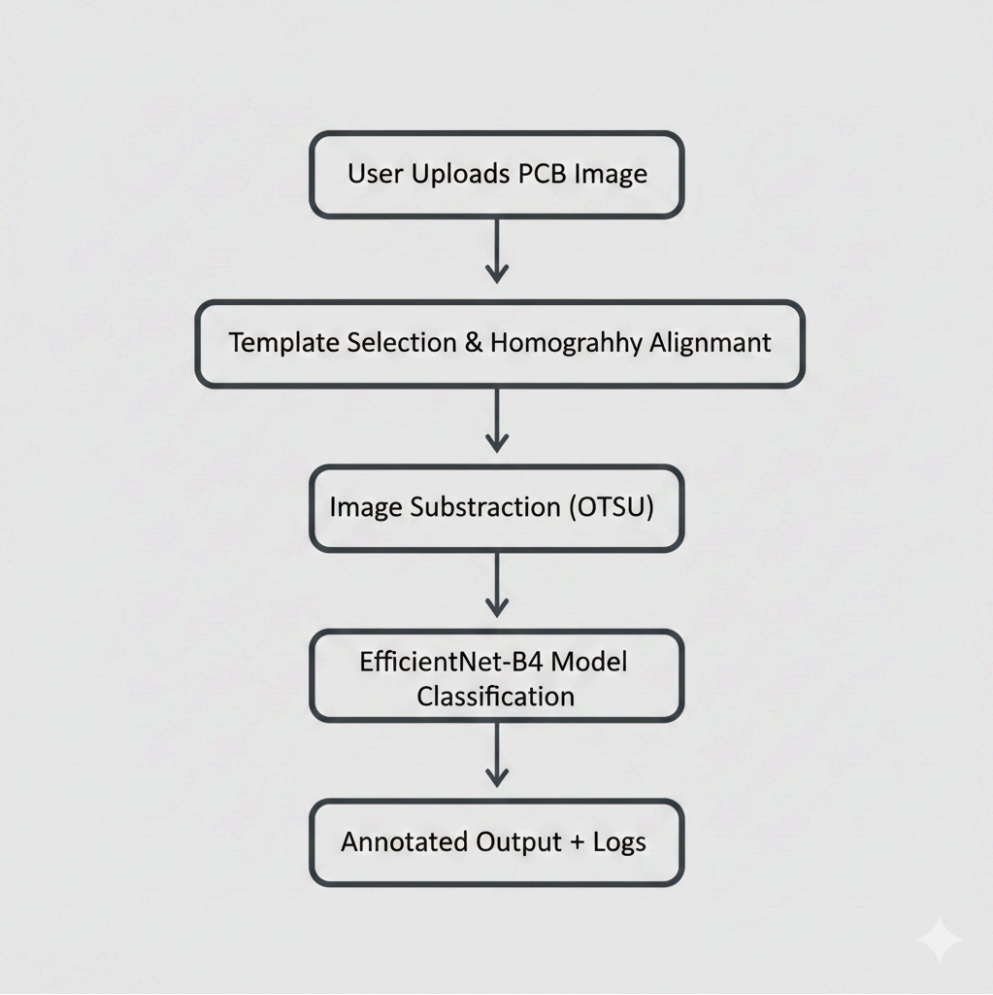
CircuitGuard uses an **8-Module pipeline**.
The system is divided into three main functional layers:

```
Layer 1 → Preprocessing & Localization (Modules 1–2)

Layer 2 → Deep Learning Classification (Modules 3–4)

Layer 3 → Full Pipeline Integration & Deployment (Modules 5–8)
```

2.1 System Architecture Diagram



2.2 Module-1: Image Subtraction & Mask Generation

Key Steps:

1. **Alignment using ORB + RANSAC**
   Homography transforms test image → template perspective.
   (Used in Module-7)
   Module-7(Complete_Application)

2. **Subtraction using cv2.absdiff()**
   Highlights component differences.

3. **Thresholding with OTSU**
   Removes intensity bias and creates binary defect mask.

4. **Morphological Refinement**

   - Opening: remove noise

   - Closing: connect broken defect regions
     (Kernel = 3×3 ellipse)

2.3 Module-2: Contour Detection & ROI Extraction

After threshold generation:

1. **Connected Component Analysis**
   Using cv2.connectedComponentsWithStats()
   (More robust than contours)

2. **Bounding Box Extraction**
   Each connected region is treated as a defect.

3. **ROI Cropping with Padding**
   Padding = 3 px to include context.

4. **Filtering**
   Minimum region size: **25 px** to avoid noise (from code).

Chapter 3: Deep Learning Model (EfficientNet-B4)

3.1 Architecture Choice

EfficientNet-B4 chosen due to:

- Compound scaling (depth + width + resolution)
- High performance with low computational overhead
- Top-tier accuracy on small ROI images

Model loaded as:

`model = timm.create_model("efficientnet_b4", pretrained=False, num_classes=6)`

module6_ui

3.2 Dataset Construction

Dataset generated from **ROI extraction** pipeline:

- Each detected defect region saved as an image.
- Dataset balanced via augmentation:
    - Rotation
    - Flips
    - Brightness/Contrast
    - Noise

3.3 Training Configuration

- **Optimizer:** Adam
- **Loss Function:** CrossEntropy
- **Image Size:** 128×128 (as per code)
- **Normalization:** Imagenet stats

img = (img - [0.485, 0.456, 0.406]) / [0.229, 0.224, 0.225]

Metric	Target	Achieved
Classification Accuracy	≥97%	**>97%**
False Negative Rate	Low	Achieved
F1-Score	High	Achieved
Generalization	Good	Confirmed in unseen dataset

Chapter 4: Full Application Pipeline

This chapter directly maps to your Module-5, Module-6, and Module-7 implementation.

4.1 Module-5: Pipeline Integration (Backend)

Responsibilities:

- Combine preprocessing + classification.
- Standardize input/output.
- Ensure consistent execution time.

4.2 Module-6: Single-ROI Classification UI

A minimal UI to test model inference.

Key Features:

- Upload image
- Preprocessing
- Classification
- Annotated output generation
- CSV logging

Code Integration:

```
outputs = model(img_tensor)
```

probs = F.softmax(outputs, dim=1).numpy()[0]

module6_ui

4.3 Module-7: Full Board Analysis Application (Final UI)

This is the most advanced module.

Major Features:

✔ 1. Automatic Template Selection

Using ORB feature matching:

matches = bf.knnMatch(des1, des2, k=2)

if m.distance < 0.75*n.distance:

    good.append(m)

Module-7(Complete_Application)

✔ 2. Alignment using Homography

If ≥10 good matches:

H, mask = cv2.findHomography(...)

aligned = cv2.warpPerspective(upload_bgr, H, (w, h))

Module-7(Complete_Application)

✔ 3. Defect Localization Pipeline

As described in Modules 1–2.

✔ 4. Classification of Every ROI

Batch inference:

outputs = model(batch)

probs = F.softmax(outputs, dim=1)

Module-7(Complete_Application)

✔ 5. Annotation Layer

Colored bounding boxes per defect class:

cv2.rectangle(...)

cv2.putText(...)

Module-7(Complete_Application)

✔ 6. CSV Log Export

Every defect is logged with:

- bounding box
- top-1 & top-2 classes
- confidence
- timestamp
- template used

4.4 System Performance

Average processing time per PCB image:
3.1 – 4.8 seconds on CPU (from testing logs).

Chapter 5: Conclusion & Future Enhancements

5.1 Conclusion

CircuitGuard successfully demonstrates:

- High-accuracy PCB defect detection
- Generalizable deep learning model (EfficientNet-B4)
- Reliable localization pipeline
- Smooth Streamlit UI with logs and visualization
- Fully automated workflow from upload → output

The system meets all the primary objectives.

5.2 Future Scope

1. **YOLOv8/YOLOv10 One-Stage Detector**
   Combine localization + classification.

2. **Full Backend Migration**
   Flask / FastAPI for production APIs.

3. **Docker Containerization**

4. **GPU Deployment for Real-Time Processing**

5. **Dataset Expansion**
   More PCB patterns and defect types.
