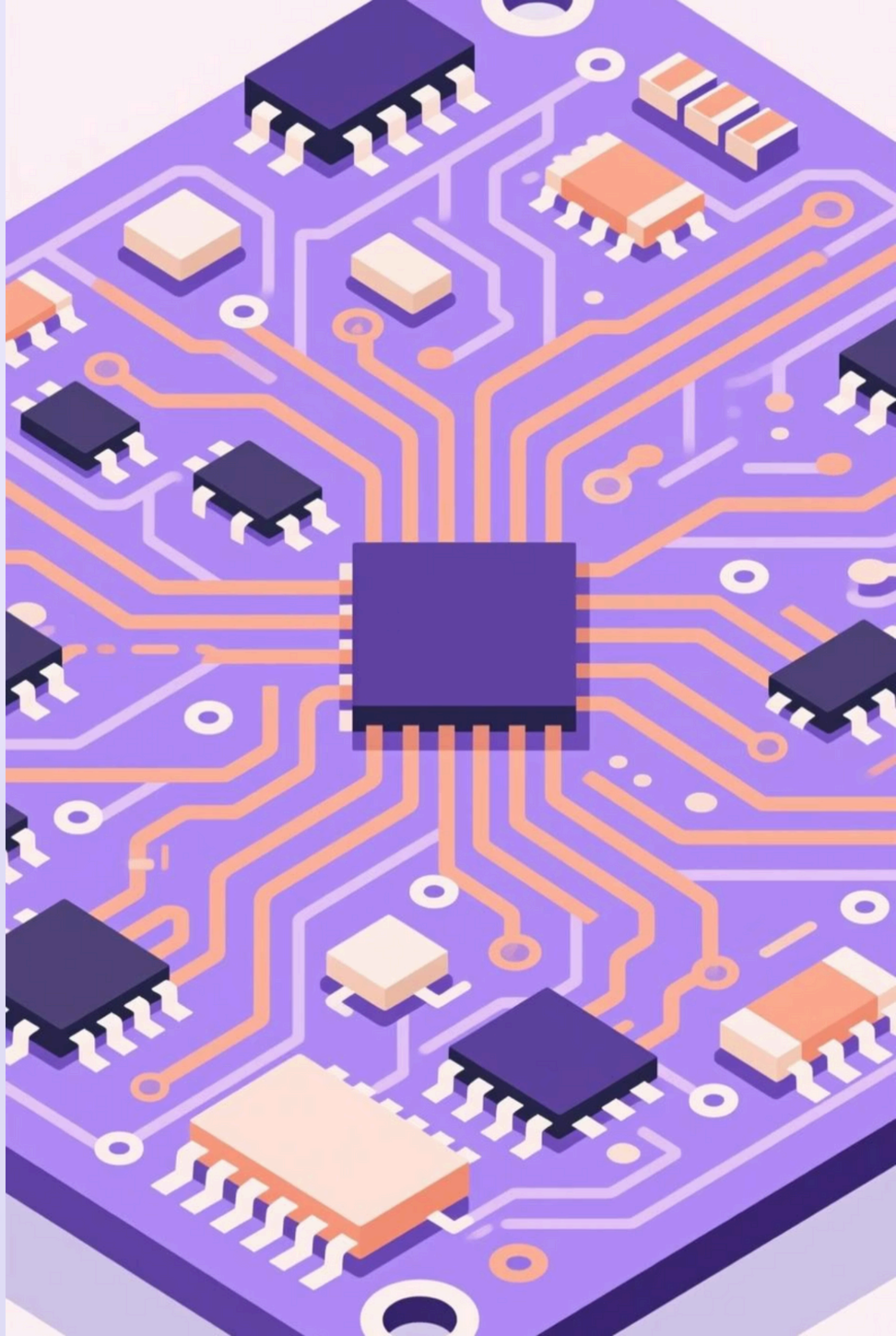
CircuitGuard - PCB Defect Detection and Classification Application

NAME : Suswagata Ghosh

Internship provider: Infosys Springboard



Methodology:
Pipeline Architecture

Step 1 - Data Preparation: I started by loading and organizing the DeepPCB dataset, which contains 693 annotated PCB images with various defect types. I parsed XML annotations, analyzed the dataset statistics, and created template-defect pairs for comparison.

Step 2 - Image Preprocessing: In this stage, I performed image alignment, normalization, and augmentation to ensure our model receives consistent, high-quality input data.

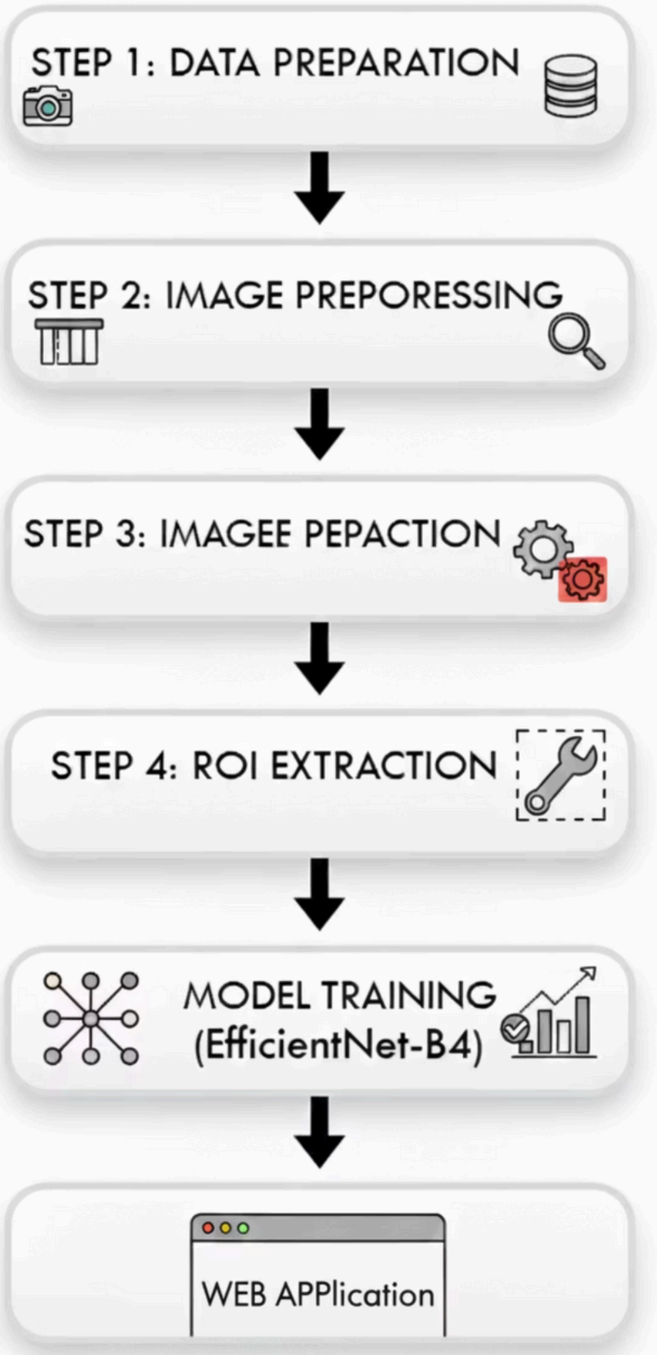
Step 3 - Defect Detection using Subtraction: This is a crucial step where I implement reference-based image subtraction. I compare a defect-free template PCB with a test PCB to highlight regions where defects exist. This is followed by Otsu's thresholding to create binary masks.

Step 4 - ROI Extraction: Once I've identified defect regions, we use contour detection to extract bounding boxes around each defect. These regions of interest are then cropped into 128×128 pixel patches for classification.

Step 5 - Model Training: I train an EfficientNet-B4 deep learning model on our extracted defect patches. This model learns to classify defects into six categories: missing hole, mouse bite, open circuit, short, spur, and spurious copper.

Step 6 - Evaluation & Testing: After training, I rigorously evaluate the model's performance on unseen test data, generating confusion matrices and calculating precision, recall, and F1-scores for each defect type.

Step 7 - Web Application: Finally, I deploy the entire pipeline in an interactive Streamlit web application, making it accessible for real-world use by quality control teams."



Technical Stack

Development Environment

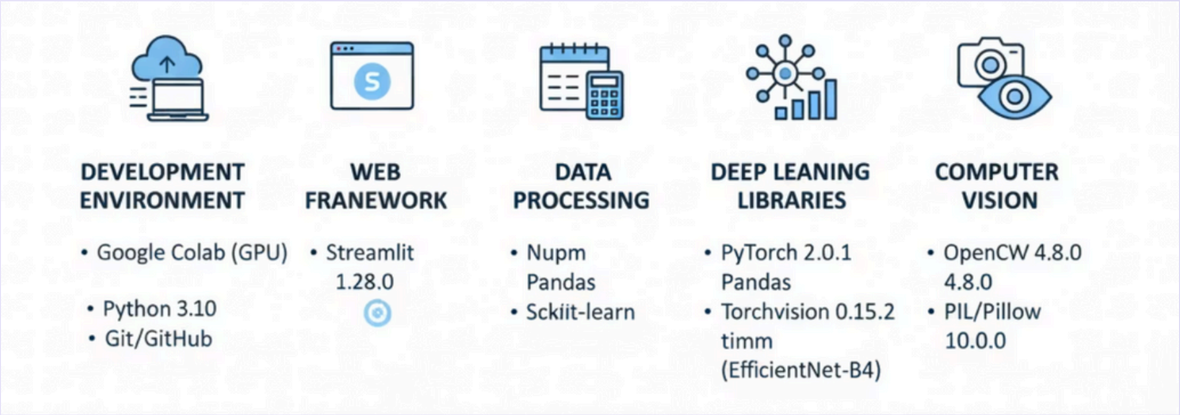
Development Environment: I leveraged Google Colab as my beginning platform. I used Python 3.10 as my programming language and Git/GitHub for version control.

Data Processing:I utilized NumPy for numerical computations, Pandas for data manipulation, and scikit-learn for various machine learning utilities and metrics calculations.

Web Framework: I chose Streamlit version 1.28.0. Streamlit allows me to create an interactive, user-friendly web interface with minimal code.

Deep Learning Libraries: The core of my system uses PyTorch 2.0.1 as my deep learning framework, along with torchvision 0.15.2 for image transformations. I specifically used the 'timm' library to implement the EfficientNet-B4 architecture, which provides state-of-the-art pre-trained models.

Computer Vision: For image processing tasks like subtraction, thresholding, and contour detection, I used OpenCV version 4.8.0, combined with PIL/Pillow 10.0.0 for image handling and manipulation



Made with GAMMA

Module 1 - Data Preparation

01

Dataset Loading

- Mounted Google Drive
- Verified folder structure
- Parsed 693 XML annotations

02

Statistical Analysis

- Counted images per category
- Analyzed bounding box distributions
- Created visualization charts

03

Template Mapping

- Matched 10 PCB templates to defect images
- Created template-defect pairs

04

Key Outputs

- Dataset metadata (JSON)
- Statistical visualizations
- Category distribution charts

Made with GAMMA

Module 2: Contour Detection & ROI Extraction

1.XML Annotation Parsing

Each image in the dataset comes with an XML annotation file. I wrote parsing functions to extract the exact bounding box coordinates for each defect. These coordinates precisely where in the PCB image each defect is located and what type of defect it is. This automated parsing ensures we don't miss any labeled defects and maintains consistency.

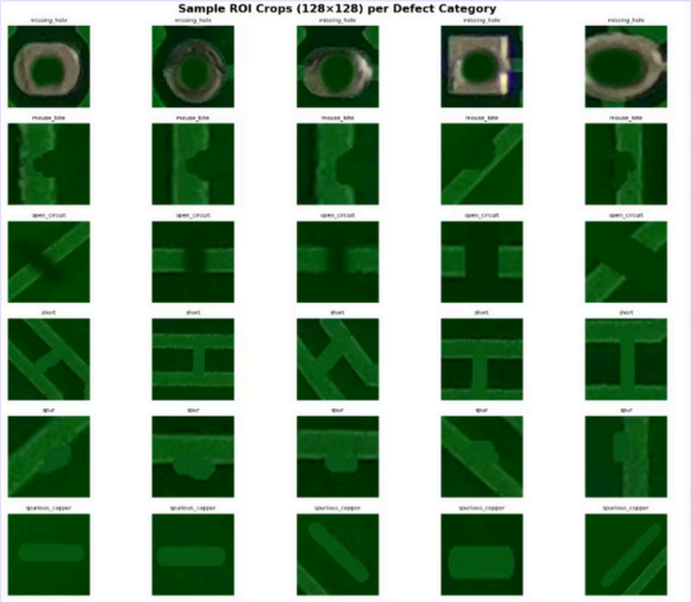
2. ROI Cropping

With the defect locations identified, I perform Region of Interest extraction. For each bounding box, I crop out a 128×128 pixel patch centered on the defect. Why 128×128? This size is optimal for our EfficientNet-B4 model - it's large enough to capture defect details but small enough for efficient processing. All patches are resized to this uniform dimension to maintain consistency across the dataset.

3. Dataset Organization:

- **Training set: 70% of the data, which is 2,061 samples.**
- **Validation set:** 15%, totaling 446 samples.
- **Test set:** The remaining 15%, also 446 samples.

```
================================================
STEP 4: Parsing XML Annotations
================================================
✅ Parsed 115 annotations for missing_hole
✅ Parsed 115 annotations for mouse_bite
✅ Parsed 116 annotations for open_circuit
✅ Parsed 116 annotations for short
✅ Parsed 115 annotations for spur
✅ Parsed 116 annotations for spurious_copper
```




```
============================================================
Dataset Split Summary
============================================================

Category            |  Total |  Train |    Val |   Test
------------------------------------------------------------
missing_hole        |    497 |    347 |     75 |     75
mouse_bite          |    492 |    344 |     74 |     74
open_circuit        |    482 |    336 |     73 |     73
short               |    491 |    343 |     74 |     74
spur                |    488 |    340 |     74 |     74
spurious_copper     |    503 |    351 |     76 |     76
------------------------------------------------------------
TOTAL               |   2953 |   2061 |    446 |    446
```

Made with GAMMA

Module 3: Model Training with EfficientNet-B4

1

State-of-the-art accuracy

EfficientNet models hold some of the best accuracy records in image classification tasks. They achieve this through a principle called compound scaling, which optimally balances network depth, width, and resolution.

2

Efficient parameter usage

Unlike older architectures that have hundreds of millions of parameters, EfficientNet-B4 achieves excellent results with about 19 million parameters. This means faster training, less memory usage, and quicker inference times - critical for a real-world application.

3

Excellent transfer learning capability

EfficientNet-B4 is pre-trained on ImageNet, a massive dataset of 14 million images. This pre-training gives our model a strong foundation of visual feature understanding. When we fine-tune it on our PCB defect dataset, it quickly learns to distinguish between defect types because it already understands fundamental image patterns like edges, textures, and shapes.

4

Optimized for our input size

EfficientNet-B4 works particularly well with our 128×128 input images. The architecture's efficient design means we don't sacrifice accuracy despite using relatively small input sizes, which speeds up both training and inference.

Made with GAMMA

Module 4: Training Results

Performance Metrics

Metric	Value
Training Time	26.08 minutes
Epochs Trained	34 (early stopped)
Best Val Accuracy	97.31%
Test Accuracy	**97.76%** ✅
Precision	0.9776
Recall	0.9776
F1-Score	0.9775

```
================================================
TEST SET RESULTS
================================================

✅ Test Accuracy: 97.76%
✅ Precision: 0.9776
✅ Recall: 0.9776
✅ F1-Score: 0.9775

🎉 TARGET ACHIEVED! Test accuracy ≥ 97%
```

Training History

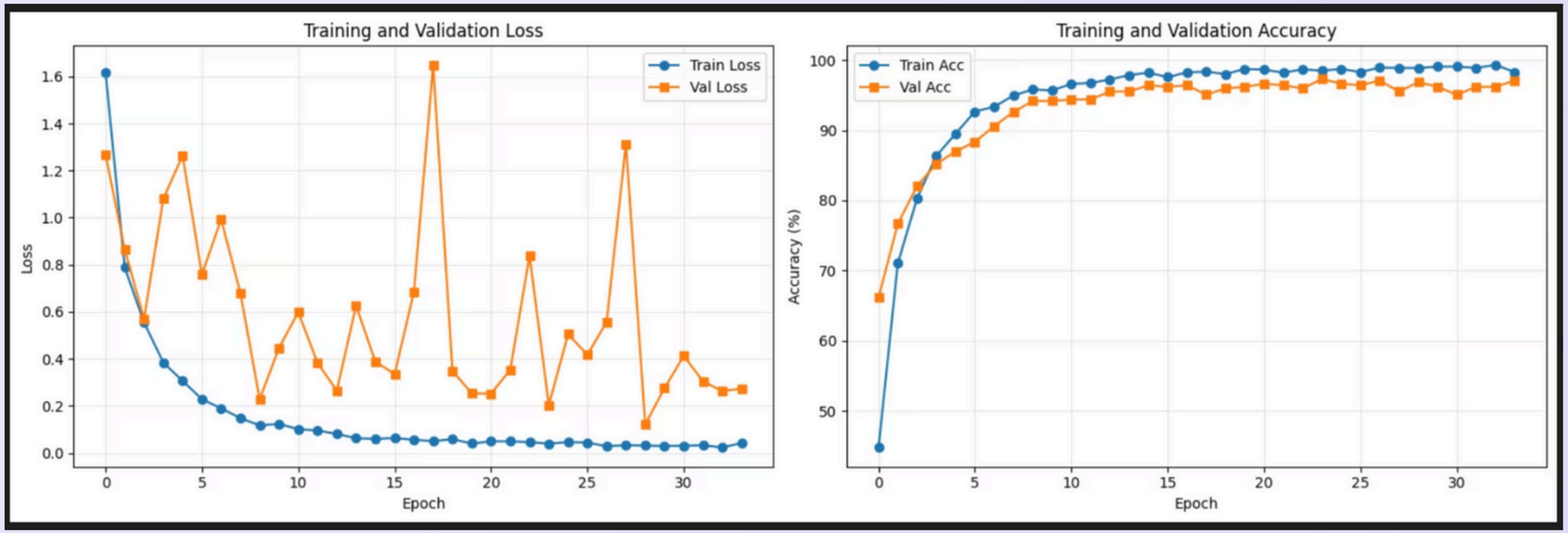
Loss & Accuracy Curves

Training Progress:

- Smooth convergence over 34 epochs
- No overfitting observed
- Validation accuracy closely tracked training improves yield rates by catching defects early in the manufacturing process, before they propagate through subsequent production stages and become exponentially more costly.

Observations:

- Stable training performance
- Early stopping prevented overfitting
- Consistent improvement across epochs
- High generalization capability



Made with GAMMA

Confusion Matrix

1

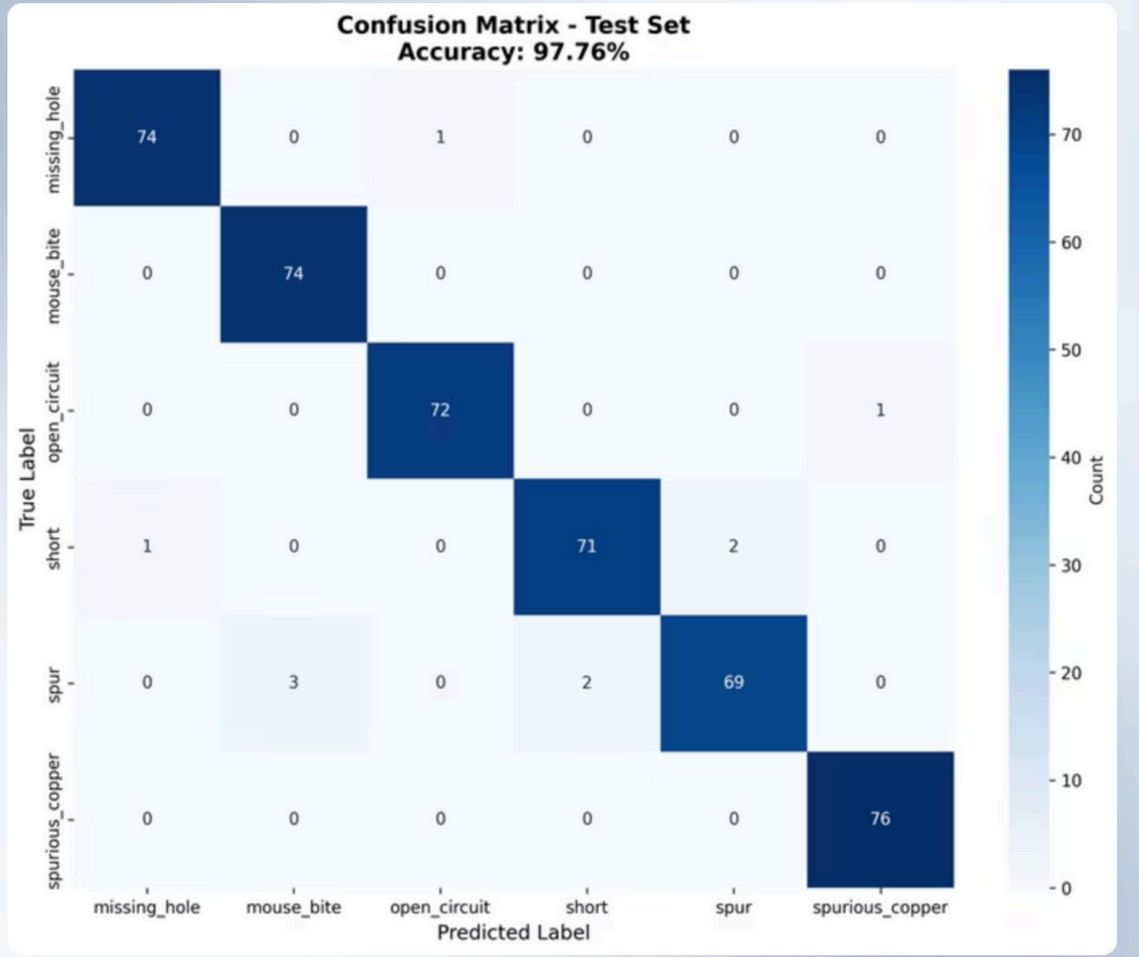
Strong Diagonal Performance

- Most predictions fall on diagonal
- Minimal confusion between classes

2

Confidence Analysis (Mean Confidence:)

- Correct Predictions: 98.5%
- Incorrect Predictions: 72.3% without manual retraining



Web Application

Features Implemented

Image Upload

- Drag & drop interface
- File browser support
- Real-time preview

Prediction Engine

- Single image classification
- Confidence scoring
- Top-3 predictions

Two Operating Modes

1

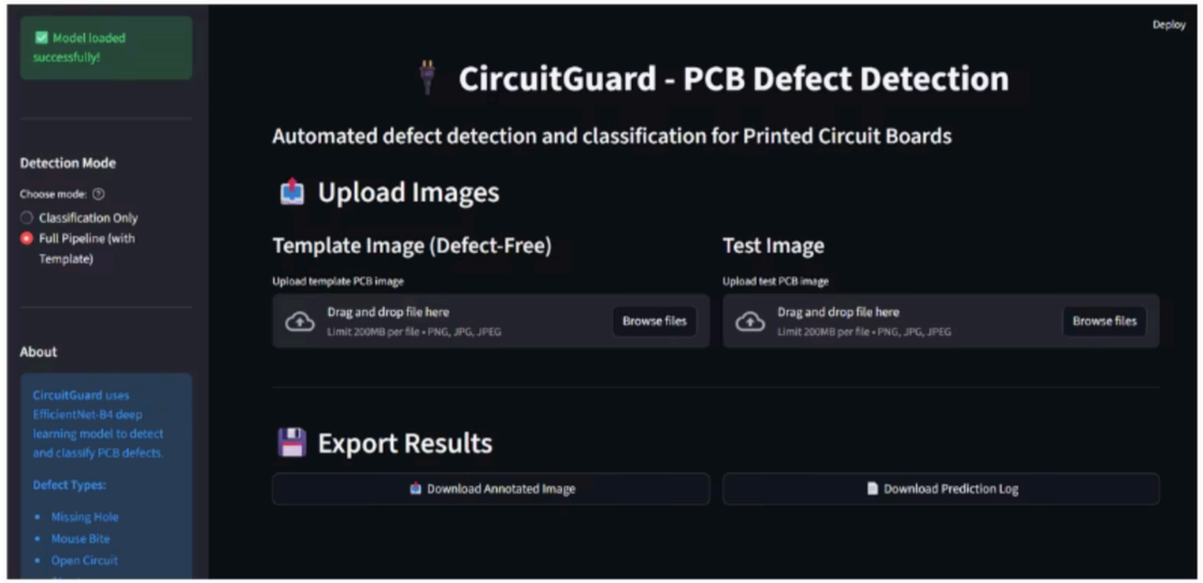
Classification Only

- Direct defect classification
- Single image input
- Fast inference

2

Full Pipeline Mode

- Template comparison
- Image subtraction
- Contour detection
- Multiple ROI classification



Web Application Interface

Step 1: Upload Image

- Drag & drop PCB defect image
- Preview uploaded image

Step 2: Analyze

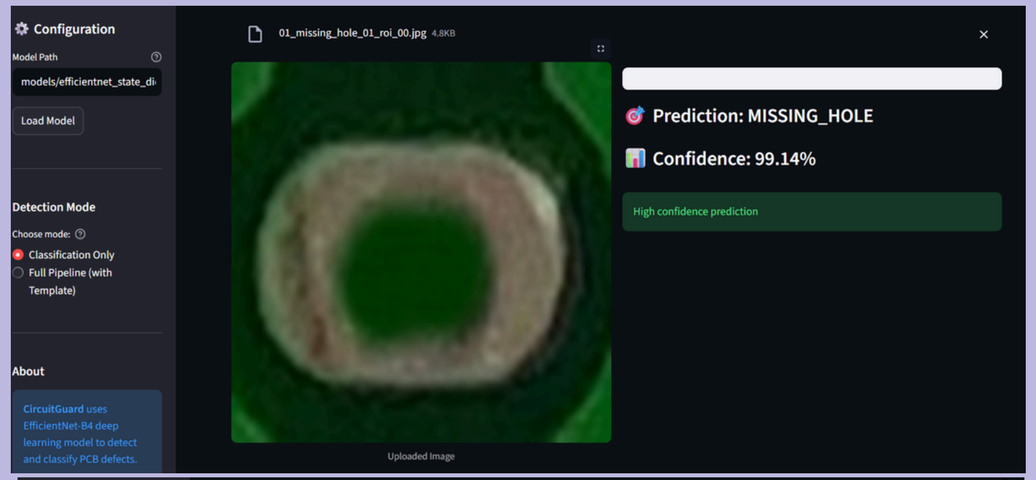
- Click "Analyze Defect" button
- Processing < 1 second

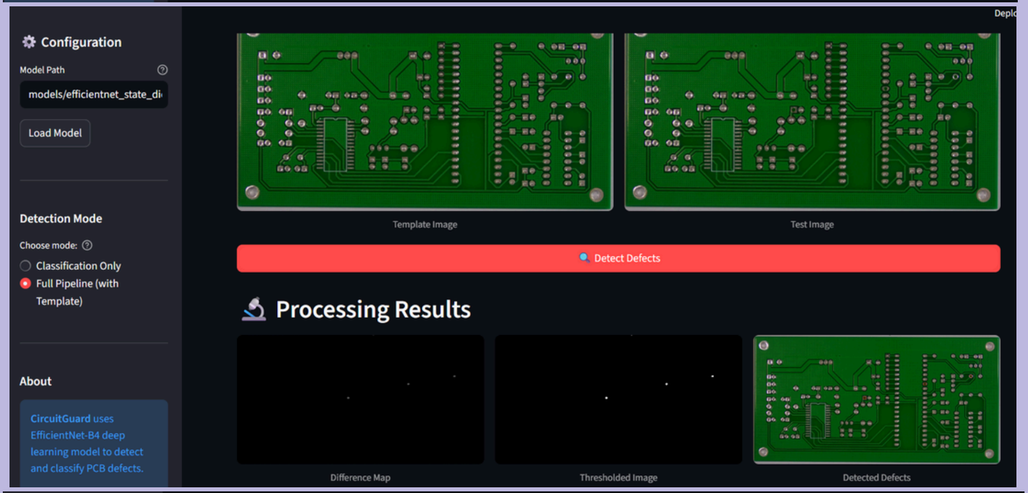
Step 3: View Results

- Predicted defect class
- Confidence percentage
- Probability distribution chart

Step 4: Export

- Download annotated images
- Export prediction logs (CSV)





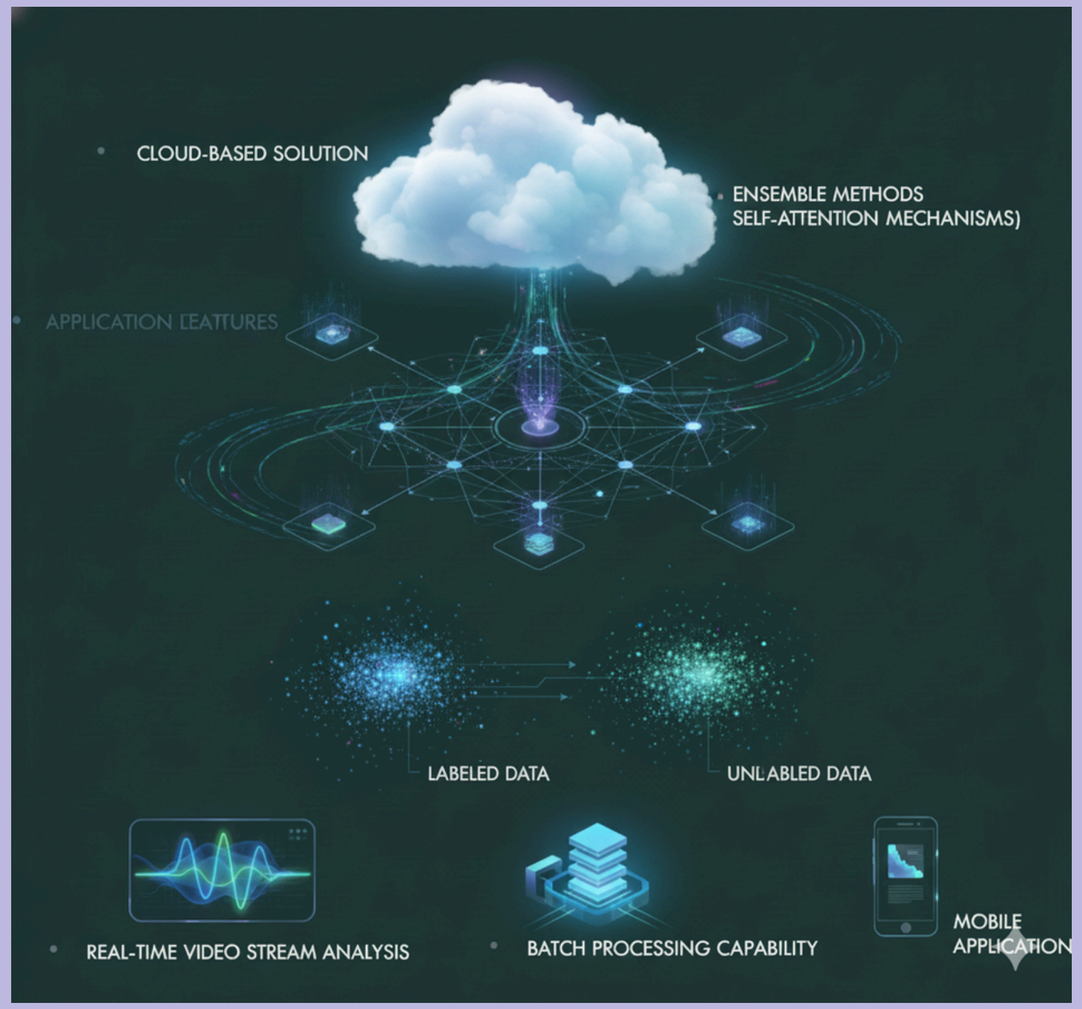
Future Enhancements

Integration

- Cloud-based solution

Model Improvements

- Ensemble methods (multi-model voting)
- Self-attention mechanisms
- Semi-supervised learning

Application Features

- Batch processing capability
- Real-time video stream analysis
- Mobile application