CircuitGuard-PCB Defect Detection and Classification Application

Project Statement:

The objective is to develop an automated defect detection and classification system for Printed Circuit Boards (PCBs) using image processing and deep learning techniques. The system will employ reference-based image subtraction, contour extraction, and CNN-based classification to identify and label defects. A fully functional frontend web application will be developed to allow users to upload PCB images and receive labeled outputs highlighting defects.

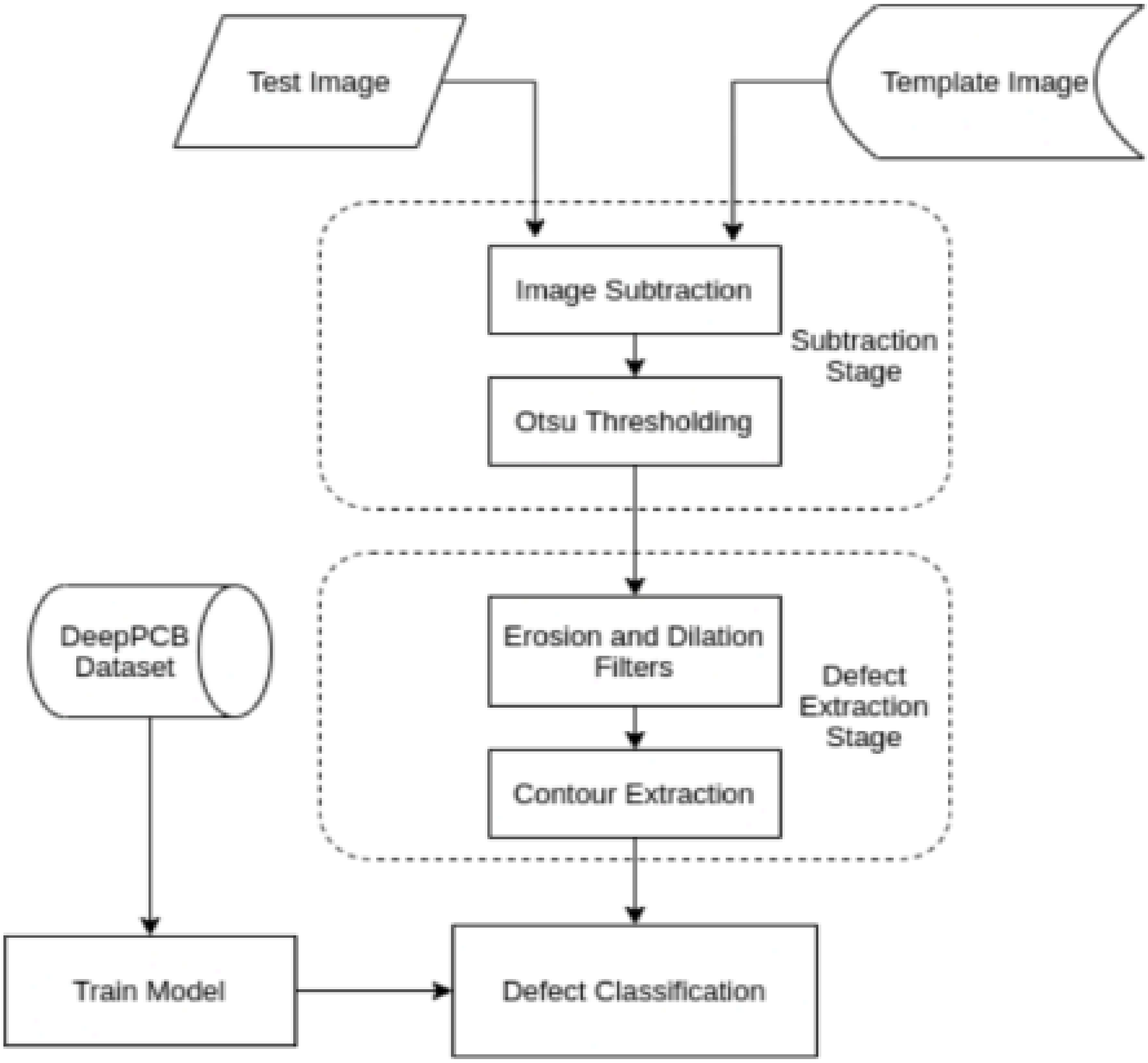
Outcomes:

● Detect and localize defects in PCB test images using comparison with defect-free templates.
● Classify detected defects into predefined categories using a trained CNN model. ● Train and evaluate EfficientNet model for robust classification.
● Build a web-based frontend for uploading images and viewing labeled outputs. ● Integrate a backend pipeline that processes input images and returns annotated results. ● Export annotated outputs and logs for documentation or analysis.

Modules to be implemented:

- Image preprocessing and subtraction using OpenCV
- Contour extraction and ROI segmentation
- EfficientNet-based image classification using PyTorch
- Frontend interface for image upload and result display
- Backend pipeline for processing and model inference
- Testing, Evaluation & Optimization
- Final Presentation & Documentation.

Dataset:
https://www.dropbox.com/scl/fi/4vrtqn7t001yl41oucflu/PCB_DATASET.zip?rlkey=pghz15q2bsg205wynjwsj2c3n&e=1&dl=0



Milestone 1: Dataset Preparation and Image Processing (Weeks 1–2)

Module 1: Dataset Setup and Image Subtraction

Tasks:

- Set up and inspect DeepPCB dataset.
- Align and preprocess image pairs (template and test).
- Apply image subtraction to obtain defect difference maps.
- Use thresholding (Otsu’s method) and filters to highlight defect regions.

Deliverables:

- Cleaned and aligned dataset
- Subtraction and thresholding script
- Sample defect-highlighted images

Evaluation:

- Accurate defect mask generation
- Proper image alignment and subtraction clarity

Module 2: Contour Detection and ROI Extraction

Tasks:

- Use OpenCV to detect contours of defects.
- Extract bounding boxes and crop individual defect regions.
- Label defect ROIs for model training.

Deliverables:

- ROI extraction pipeline
- Cropped and labeled defect samples
- Visualization of defect contours

Evaluation:

- Precision of ROI detection and bounding box accuracy

Milestone 2: Model Training and Evaluation (Weeks 3–4)

Module 3: Model Training with EfficientNet

Tasks:

- Implement EfficientNet-B4 using PyTorch.
- Preprocess and augment defect images (128x128 size). ●

Train model using Adam optimizer and cross-entropy loss.

Deliverables:

- Trained EfficientNet model
- Accuracy and loss metrics
- Evaluation plots and confusion matrix

Evaluation:

- ≥ 97% classification accuracy on test set
- Stable and repeatable training performance

Module 4: Evaluation and Prediction Testing

Tasks:

- Test model on new unseen test images.
- Run inference pipeline and validate predictions.
- Compare results against annotated ground truth.

Deliverables:

- Annotated output test images
- Final evaluation report with metrics

Evaluation:

- Prediction match rate with annotated truth
- Low false positive/negative rate

Milestone 3: Frontend and Backend Integration (Weeks 5–6)

Module 5: Web UI for Image Upload and Visualization

Tasks:

- Build frontend using Streamlit or HTML, CSS, JS.

- Add upload fields for template and test images.
- Display output images with bounding boxes and labels.

Deliverables:

- app.py frontend script
- Real-time UI for defect prediction
- Labeled image preview with results

Evaluation:

- Usable, responsive UI with no upload/rendering issues

Module 6: Backend Pipeline for Image Inference

Tasks:

- Modularize image processing and model inference functions.
- Integrate EfficientNet model checkpoint for prediction. ●
  Connect backend to frontend upload inputs.

Deliverables:

- Backend logic with full prediction pipeline
- Return annotated images and logs

Evaluation:

- Smooth backend function with UI inputs
- Output generated with less lag

Milestone 4: Finalization and Delivery (Weeks 7–8)

Module 7: Testing, Evaluation, and Exporting Results

Tasks:

- Add option to download labeled image and prediction log
- Test application on multiple image pairs
- Optimize pipeline for speed and performance

Deliverables:

- Finalized web app with export button
- Annotated output images and CSV logs

Evaluation:

- Fully functional UI with successful export feature

Module 8: Documentation and Final Presentation

Tasks:

- Prepare technical documentation and README ●

Write user guide for frontend and backend usage ●
Create demo video or slide deck for presentation

Deliverables:

- Final documentation PDF
- GitHub repo with organized folders and scripts
- Recorded walkthrough and slides

Evaluation:

- Clear and complete documentation
- Demo-ready project presentation

Evaluation Criteria:

Milestone	Focus Area	Metric / Evaluation Method	Target/Goal
Milestone 1	Image Processing	Mask quality, bounding box accuracy	Detect all key defect areas
Milestone 2	Model Performance	Accuracy, Confusion Matrix	≥ 97% test accuracy
Milestone 3	UI Integration	Upload-to-output time	≤ 5 seconds per image
Milestone 4	System Finalization	Export functionality, documentation	Fully working deliverables
Overall	Project Delivery	GitHub + UI + Documentation	Professional-quality outcome

Tech Stack:

Area	Tools / Libraries
Image Ops	OpenCV, Numpy
Model	PyTorch, timm, EfficientNet-B4
Dataset	DeepPCB
Frontend	Streamlit /HTML, CSS,
Backend	Python, Modularized Inference
Evaluation	Accuracy, Loss, Confusion Matrix
Exporting	CSV, Annotated Image, PDF (opt.)