PCB Defect Detection System for Quality Control

Dawson Burgess
Computer Science Department
University of Idaho, Moscow, ID
Moscow, ID, United States
burg1648@vandals.uidaho.edu dawsonhburgess@gmail.com

***Abstract*—Printed circuit board (PCB) defects like missing components, broken traces, and incorrect soldering present problems to the reliability and performance of electronic devices. Traditional inspection methods such as manual checks and specialized hardware are costly, time-consuming, and prone to human error. This project presents a vision-based PCB defect detection system that makes use of deep learning techniques. The specific methods used were Convolutional Neural Networks (CNNs) and several more "traditional" models. This system includes data preprocessing and labeled defect annotations to train and evaluate models. The models demonstrated excellent classification performance across multiple defect types. Compared to traditional machine learning models, the CNN approach delivers improved accuracy and more balanced performance. Results indicate that a deeper CNN architecture achieves up to 96% accuracy which performs better than both the baseline CNN and traditional methods. These findings highlight the effectiveness of end-to-end learned feature extraction in detecting subtle PCB anomalies. This paper and others help lay the path for future enhancements such as real-time inspection and advanced augmentation strategies.**

Keywords—Printed Circuit Board (PCB), Defect Detection, Computer Vision, Deep Learning, Convolutional Neural Networks (CNN), Machine Learning, Quality Control.

I. Introduction

Printed circuit boards (PCBs) serve as the basis of nearly every electronic device from consumer electronics to industrial control systems. The quality and reliability of PCBs is very important because even minor defects (missing holes, broken traces, soldering issues, or misplaced components) can lead to product failure and malfunction. PCB inspection has relied on manual visual checks or dedicated inspection hardware. Both of these methods can be costly, labor-intensive, and prone to human error. There is a need for an automated and accurate vision-based detection system capable of identifying a broad range of faults.

The challenge of PCB defect detection is accurately distinguishing subtle anomalies from normal board patterns. Variations in lighting, board design, component placement, and imaging conditions further complicate this problem. Attempts to tackle PCB defect detection have used image processing algorithm such as edge detection, template matching, and morphological operations to enhance images and highlight patterns before classification [2] [6]. These approaches successfully identified certain defects, but their performance deteriorated in complex scenarios.

Recent research has shifted toward data-driven methods involving machine learning (ML) and deep learning. ML pipelines often combine hand picked features with classifiers such as Support Vector Machines (SVMs) or Random Forests [3]. These methods perform better than manual or rule-based inspections, but then they tend to struggle when defect patterns become nuanced. Deep learning-based models like Convolutional Neural Networks (CNNs) learn hierarchical features directly from raw image data which in turn improves robustness and versatility [1] [4] [5].

An increasing number of studies has demonstrated the potential of deep learning to enhance PCB defect detection. Zhang et al. proposed deep feature learning for bare PCB defects. They showcased improved detection rates over traditional approaches [5]. Chen et al. conducted a comprehensive review of deep learning-based PCB defect detection and exhibited how neural networks can adapt to numerous defect types and manufacturing conditions [7]. Zhou et al. also reviewed vision-based defect detection techniques, placing emphasis on the evolution from classical image processing to modern deep learning approaches [4]. These studies all highlight that CNNs and related architectures tend to exceed the capabilities of traditional methods.

This project will build on these insights to develop a open-source vision-based PCB defect detection system. The approach involves preparing a labeled dataset of PCB images, applying preprocessing steps to boost defect visibility, and then employing classification models to identify defects. By comparing deep learning methods against classical ML models, the goal is to illustrate the increased accuracy of learned feature extraction. Although the final scope focuses primarily on end-to-end CNN models, this work lays the groundwork for future enhancements, such as integrating classical techniques for initial region-of-interest detection or extending the system to real-time inspection using live video feeds.

This research contributes to the evolving landscape of automated PCB quality control, aligning with recent advancements in ML frameworks [1], [3] and bridging gaps identified in prior literature [2], [4], [5]–[7]. The findings underscore the practical and theoretical importance of adopting deep learning strategies for reliable, high-performance PCB defect detection.

II. Materials and Methods

A. Data Acquisition and Preparation

This project utilized a dataset of PCBs containing various defects. This dataset is named the "PCB Defect Dataset," and is readily available on Kaggle through the link in the references section. The dataset included both high-resolution PCB images

and their corresponding annotation files. Each annotation described the location (bounding box coordinates) and type (class label) of one or more defects on the board.

To transform the data into a more manageable form, a Python script was developed to convert the data into a standard format. This script employed *xml.etree.ElementTree* to parse each XML file. This process extracted and reformatted relevant attributes such as the file name, defect class, bounding box coordinates (xmin, xmax, ymin, ymax), and the original image dimensions. The extracted information was consolidated into a pandas *DataFrame*. The advantage of using this new tabular representation is:

1) Unified Access: It facilitates quick inspection of dataset statistics and permits identification of class imbalances, distribution of bounding box sizes, and any highlighting of anomalies in the annotations (e.g. missing values or overlapping bounding boxes).

2) Reproducibility: Storing parsed data as a CSV file ensured that subsequent model training steps would not need to re-parse XML files. This enables an easily reproducible and streamlined process for running the rest of the code.

B. Data Cleaning and Preprocessing

Initial examination of the DataFrame revealed missing or inconsistent annotations like bounding boxes with zero width/height or mislabeled classes. Rows containing such figures were removed to maintain data integrity. Any annotations referencing image files that could not be loaded were also discarded. This cleaning process made sure that only well-formed annotations remained.

For training CNN-based models, consistent input dimensions and normalized pixel values were needed. Each defect region was extracted from its corresponding image using the provided bounding box coordinates. These images were resized to a fixed size of 224x224 pixels. Resizing standardized input dimensions and also improved batching and memory usage during training.

Normalization was performed by scaling pixel intensities to the [0,1] range. This step helped training by ensuring that the input features had a similar magnitude. Data augmentation (e.g. random flips, slight rotations) was briefly considered, but initial trials demonstrated marginal improvements at best. The structure and subtle differences in PCB patterns meant that heavy augmentation could distort the defect patterns.

C. Dataset Structure and Splitting

After cleaning and preprocessing, the dataset was split into training and testing sets at an 80/20 ratio. The training set was used to fit model weights and the test set provided an unbiased evaluation of model performance on unseen data. An even stratification was not strictly enforced because the dataset did not exhibit severe class imbalance. Care was taken to ensure that each class was reasonably represented in both splits.

D. Initial Modeling Approaches

The project began with classical machine learning techniques. These methods involved classifiers like Support Vector Machines (SVMs), Random Forests, and Logistic Regression. While these classical pipelines provided a baseline, their performance was not up to the desired standarrd due to the complexity and subtlety of PCB defects.

Recognizing the limitations of these methods, the focus shifted to deep learning approaches. The first deep learning experiments were implemented using PyTorch. Custom Dataset classes and DataLoaders were created to handle the on-the-fly loading and preprocessing of images. PyTorch offered great flexibility and control over the training loop and model design, but it became clear that rapid experimentation required a more streamlined framework.

E. Transition to Keras/TensorFlow

Early in the project, PyTorch served as the initial deep learning framework. PyTorch offers a highly flexible platform for building complex neural networks. It features a dynamic computation graph that allows for custom training loops, straightforward debugging, and direct integration with Python control flow. This flexibility can be an advantage when developing novel architectures or experimenting with unconventional training procedures.

As the project evolved, the need for rapid prototyping, a simpler model iteration, and more streamlined experimentation workflows became desired. It was at this moment that the decision was made to transition the project with a focus on PyTorch to a focus on using Keras/TensorFlow:

1) Ease of Use and Rapid Prototyping: Keras is built on top of TensorFlow, and it provides a high-level API that makes model definition, training, and evaluation better and less error-prone. The framework's *sequential* and functional APIs eliminate much of the boilerplate code associated with custom training loops. Built-in methods like model.fit() and model.evaluate() standardize common training routines. This allows for faster iteration on model architectures, hyperparameter tuning, and preprocessing adjustments without the need for rewriting core training procedures.

2) Integration with TensorFlow's Ecosystem: By using Keras/TensorFlow, the project taps into the broad TensorFlow ecosystem. These integrated tools helped streamline the workflow. This allowed for immediate logging of training progress, more intuitive debugging of layer outputs, and convenient saving/loading of models. Such an environment reduces the overhead of needing to manage separate tools for visualization and model conversion.

3) Production and Deployment Considerations: PyTorch has mature and complex deployment pathways, but TensorFlow's ecosystem offers more out-of-the-box options for production environments. These include things such as TensorFlow Serving and compatibility with a wider range of

hardware accelerators. The project may not be at deployment stage, but knowing that the chosen framework supports efficient scaling and deployment later can be a valuable long-term advantage.

F. Advantages of PyTorch We Miss Out On

By transitioning to Keras/TensorFlow, some of PyTorch's strengths are less fully utilized:

1) Greater Flexibility in Research-Oriented Experimentation:
PyTorch's execution model and straightforward integration with Python's control flow make it easier to develop custom training loops and integrate complex logic directly into the model's forward pass. For research experiments involving dynamically changing architectures, implementing novel layers or optimizers from scratch, or building reinforcement learning pipelines—PyTorch often provides a more intuitive approach.

2) Strong Community and Research Adoption for Custom Architectures:
PyTorch is widely adopted in research settings, and many state-of-the-art models and techniques are initially released in PyTorch. Keras and TensorFlow have robust communities, but missing out on PyTorch can mean that it's sometimes more challenging to replicate the latest research model. This is not a concern for this project, but it is worth note if further, more complicated research is desired.

3) Fine-Grained Control Over Training Process: Keras/TensorFlow's high-level API is beneficial for rapid iteration, but it can obscure some lower-level training details. PyTorch encourages a more hands-on approach where every training step (forward pass, backward pass, optimization) can be customized without significant overhead. Advanced experiments like building a custom gradient accumulation scheme, implementing complex loss functions that require non-standard aggregation steps, or running highly specialized forms of model parallelism are better done in PyTorch. Like mentioned above, this was not necessary for this project, yet still important to touch base on lost opportunities.

G. Neural Network Architectures and Iterative Refinement

Various CNN-based models were explored, but only two ended up being implemented:

1) Baseline CNN: A somewhat shallow network composed of a few convolutional layers followed by max pooling and dense layers. This model served as a entry point, testing whether even a simple CNN could outperform various other ML approaches. See Table 1 for an architecture overview.

Layer (Type)	Output Shape	Number of Parameters
Conv2d (Conv2D)	(None, 224, 224, 16)	448
max_pooling2d (MaxPooling2D)	(None, 112, 112, 16)	0
conv2d_1 (Conv2D)	(None, 112, 112, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 32)	0
flatten (Flatten)	(None, 100352)	0
dense (Dense)	(None, 128)	12845184
dense_1 (Dense)	(None, 6)	774

Table 1: *Baseline CNN Model Layers, Output Shape, and Number of Parameters from using Keras Summary*

2) **Deeper CNN*:*** Building on the baseline, the deeper CNN incorporated additional convolutional blocks to learn more complex patterns. This increased complexity will idealy lead to richer feature representations and potentially improve accuracy on subtle defects. See Table 2 for the deeper CNN architecture overview.

Layer (Type)	Output Shape	Number of Parameters
conv2d_2 (Conv2D)	(None, 224, 224, 32)	896
conv2d_3 (Conv2D)	(None, 224, 224, 32)	9248
max_pooling2d_2 (MaxPooling2D)	(None, 112, 112, 32)	0
conv2d_4 (Conv2D)	(None, 112, 112, 64)	18496
conv2d_5 (Conv2D)	(None, 112, 112, 64)	36928
max_pooling2d_3 (MaxPooling2D)	(None, 56, 56, 64)	0
conv2d_6 (Conv2D)	(None, 56, 56, 128)	73856
conv2d_7 (Conv2D)	(None, 56, 56, 128)	147584
max_pooling2d_4 (MaxPooling2D)	(None, 28, 28, 128)	0
flatten_1 (Flatten)	(None, 100352)	0
dense_2 (Dense)	(None, 256)	25690368
dense_3 (Dense)	(None, 6)	1542

Table 2: *Deeper CNN Model Layers, Output Shape, and Number of Parameters from using Keras Summary*

3) **Advanced Architectures (Discarded):**

a) Inception-like Modules: Inspired by GoogLeNet/Inception architectures, these modules were introduced to capture multi-scale features in parallel.

b) Depthwise Separable Convolutions: Similar to MobileNet-type models, these layers aimed to improve efficiency and possibly learn more discriminative features with fewer parameters.

While these advanced models were tested, they offered only marginal gains (if any) over the simpler CNN variants. They required longer training times and more careful tuning of hyperparameters. Given the negligible performance

improvements, these architectures were ultimately discarded in favor of the baseline and deeper CNN models.

H. Rationale for Final Model Selection

The decision to focus on the baseline CNN and the deeper CNN model was guided by empirical evidence. Both models trained relatively quickly and demonstrated stable convergence. More complex architectures did not produce a substantial increase in accuracy, precision, or recall. As a result, from an engineering and practical perspective, simplicity and efficiency were prioritized. Choosing models that were easier to maintain, interpret, and scale was deemed beneficial.

I. Experimental Setup and Hyperparameters

All CNN models were trained using the Adam optimizer with a standard learning rate (e.g., 0.001) and a cross-entropy loss function. Batch sizes that were tested ranged from 16 to 128 and ultimately chosen based on hardware constraints (GPU memory) and training stability. The end result was a batch size of 32 for the CNN and 16 for the deep CNN. Early stopping criteria were integrated, halting training if validation accuracy failed to improve after a certain number of epochs. This never ended up happening, but was kept included for future work. The number of epochs was typically around 5 to 10, adjusted based on observed convergence patterns.

J. Evaluation and Visualization Techniques

Performance was measured using accuracy, confusion matrices, and class-wise metrics such as precision, recall, and F1-score. Confusion matrices offered insights into which classes were well-distinguished and which were frequently misclassified. Additionally, Principal Component Analysis (PCA) was applied to visualize class separability in a 2D space. This visualization helped confirm whether certain defect types were inherently more distinctive or if classes overlapped significantly, helping to explain patterns in the confusion matrices.

III. Results

Following data preprocessing and model training described previously, the final experiment focused on comparing a baseline Convolutional Neural Network (CNN), a deeper CNN architecture, and three classical machine learning (ML) approaches (Support Vector Machine, Random Forest, and Logistic Regression) on the PCB defect test set. Performance was evaluated using multiple metrics, and various visualizations were generated to provide comprehensive insights into model behavior and class-level distinctions.

A. **Model Performance Overview**

Table 3 summarizes the accuracy and macro-averaged F1-scores achieved by all models. Both CNNs outperformed the classical ML methods, with the deeper CNN emerging as the top performer (achieving a 96% accuracy). The baseline CNN followed closely at 94%, while classical methods (Random Forest, SVM, and Logistic Regression) attained accuracy scores of 91%, 90%, and 82%, respectively.

Model	Accuracy	Macro Average F1-Score
Baseline CNN	0.94	~0.94
Deeper CNN	0.96	~0.96
SVM (RBF Kernel)	0.9	~0.90
Random Forest	0.91	~0.91
Logistic Regression	0.82	~0.82

Table 3: *Comparative Performance of Models on the PCB Defect Test Set*

B. Neural Network Model Results

The baseline and deeper CNN models consistently demonstrated high precision and recall across all defect classes. Their confusion matrices (Figures 1 and 2) show that common defects such as *missing_hole* and *open_circuit* were identified with near-perfect accuracy by both models. More subtle classes like *spur* and *spurious_copper* exhibited minor misclassifications, but still maintained high precision and recall values.

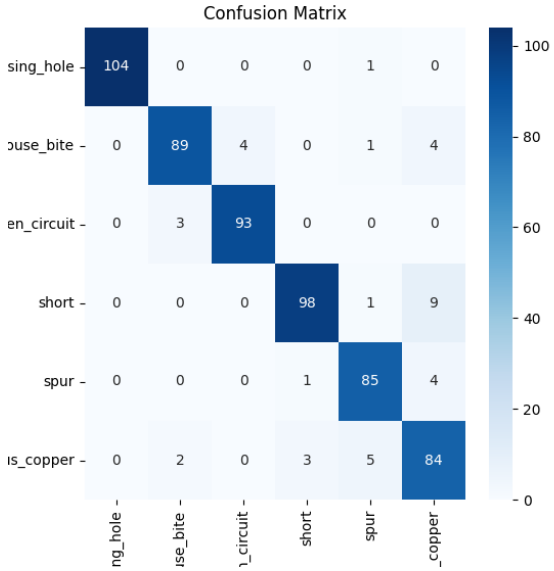

Figure 1: *Confusion Matrix for the Baseline CNN Model*

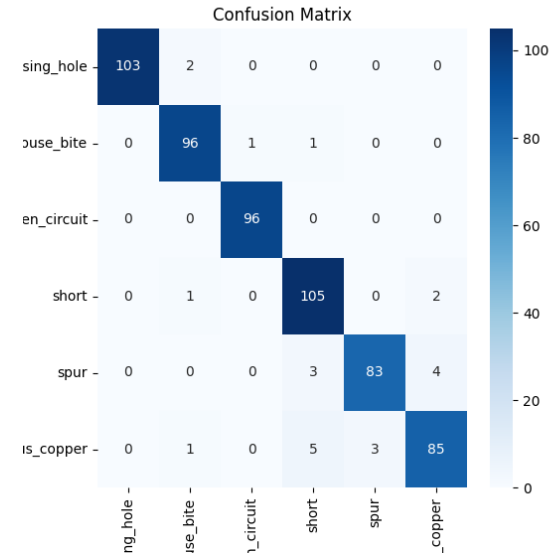

Figure 2: *Confusion Matrix for the Deeper CNN Model*

To further analyze model performance, precision-recall (PR) curves (Figure 3) and ROC curves (Figure 4) were generated for each class. These curves help illustrate how each model performs under varying classification thresholds. The deeper CNN's PR and ROC curves exhibited high areas under the curve (AUC) for all classes. This assists in confirming its robustness and class-separability. Even for the more challenging classes, the PR curves showed strong precision at a broad range of recalls and the ROC curves approached the top-left corner (low false-positive rate).

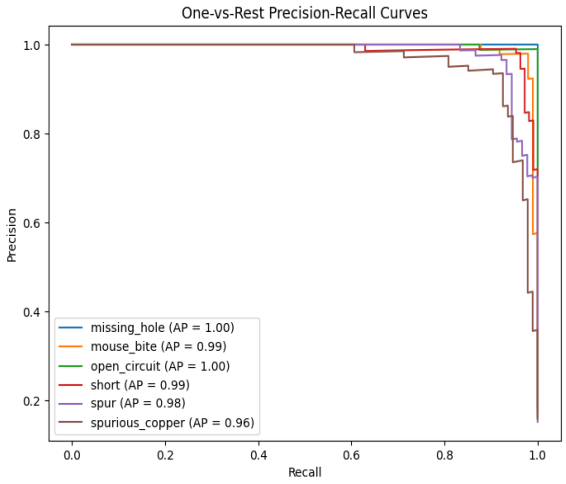

Figure 3: *Precision-Recall Curves for the Deeper CNN Model*

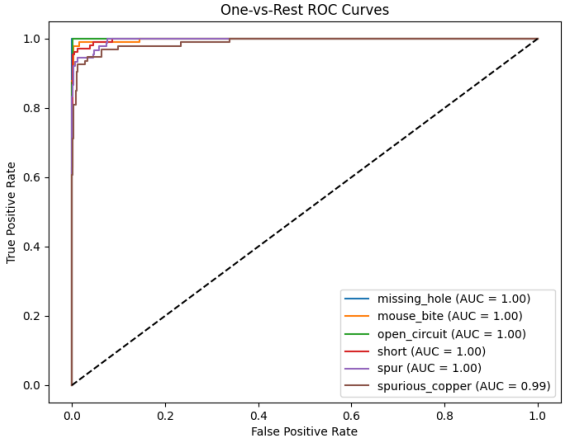

Figure 4: *ROC Curves for the Deeper CNN Model*

The training history plots for both CNN models (Figure 5 and 6) revealed stable and smooth convergence over the course of a few epochs. Both models reached their peak performance within approximately 5–10 epochs, with minimal overfitting observed. There was strange behaviour at the end of the deep CNN, indicating some overfitting, but this was not always the case. The baseline CNN plateaued slightly earlier, while the deeper CNN continued to improve marginally for a longer period, aligning with its higher final accuracy (outside of the previously mentioned dip).

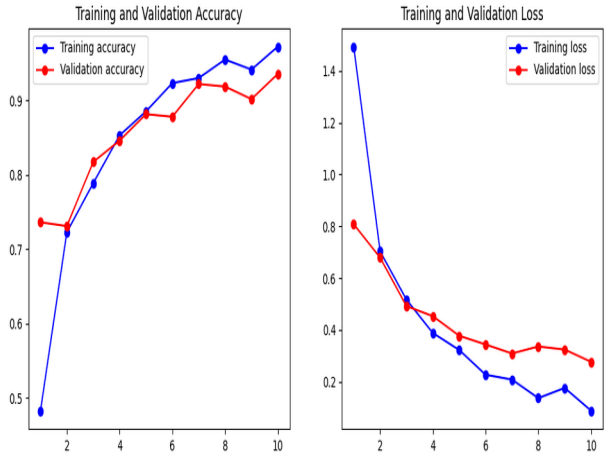

Figure 5: *Training History (Accuracy and Loss) for Baseline CNN Model*

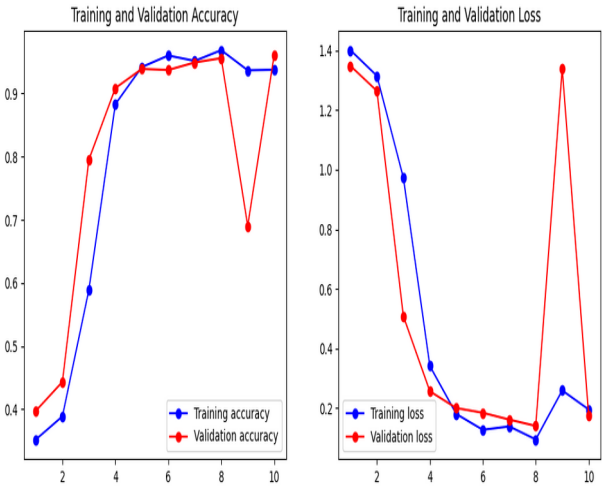

Figure 6: *Training History (Accuracy and Loss) for Deeper CNN Model*

C. Classical ML Model Comparisons

The other ML models provided baseline points of reference. Their confusion matrices (Figures 7, 8. and 9 show Random Forest, SVM, and Linear Regression) indicated a generally good performance on well-defined classes like *missing_hole* and *open_circuit*. There seemed to be more confusion on visually similar defects (e.g., *spur* vs. *spurious_copper*). The Random Forest outperformed the SVM and Logistic Regression, but none matched the CNNs' comprehensive class-level performance.

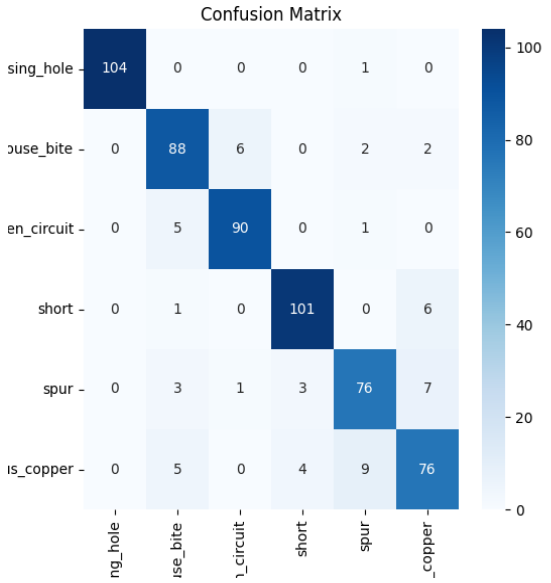

Figure 7: *Confusion Matrix for the Random Forest Model*

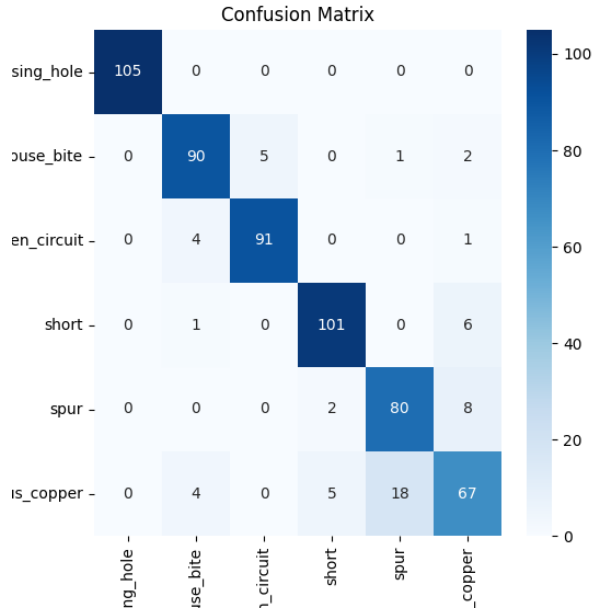

Figure 8: *Confusion Matrix for the SVM Model*

Logistic Regression (Figure 9) struggled the most, reflecting the limitations of linear decision boundaries in differentiating subtle defect patterns. Most classes remained well-classified, but performance for challenging classes dropped quite significantl. This is indicated by more off-diagonal counts in the confusion matrix with up to 18 mistakes.

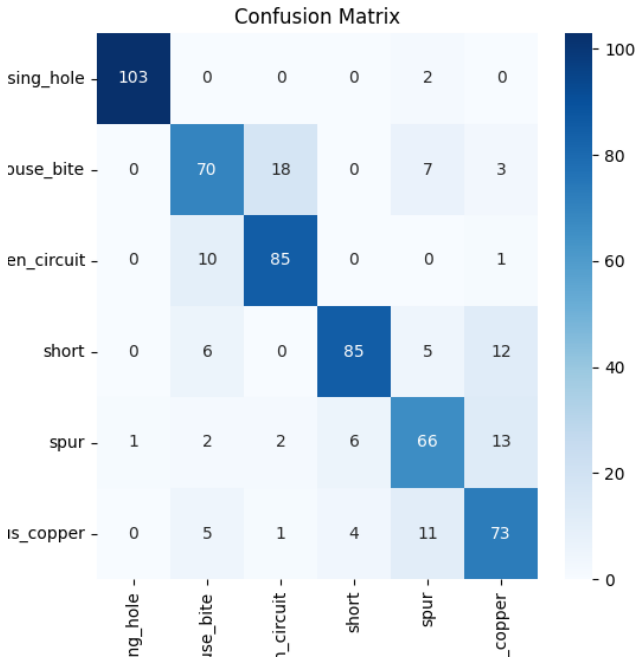

Figure 9: *Confusion Matrix for the Logistic Regression Model*

D. *Feature Space Visualization*

To understand feature separability learned by the models, a PCA-based visualization was generated (Figure 10). This 2D projection revealed that classes such as *missing_hole* and *open_circuit* formed distinct clusters, correlating with their high classification scores. Classes like *spur* and *spurious_copper* seem to be more intermixed, supporting the finding that these classes are inherently more challenging to distinguish. This makes sense looking at the visual examples of faults like *missing_hole* (arguably the easiest to classify). The PCA plot thus provides a qualitative explanation for the misclassifications observed in both CNN and classical ML models.

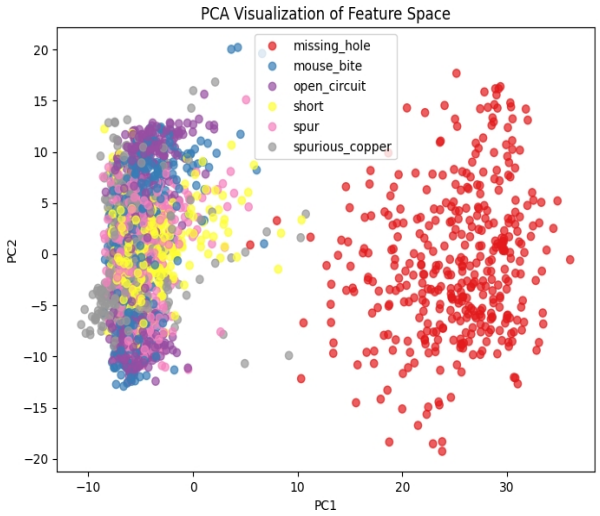

Figure 10: *PCA Visualization of the Sklearn Models*

E. Miss-classification Analysis

Both the baseline and deeper CNN models achieved high overall accuracy, and a closer look at the misclassified samples offers additional insight into the remaining challenges that exist in PCB defect detection. Certain classes with subtle visual distinctions sometimes led the models to produce incorrect predictions. For instance, defects that closely resembled each other like a "spur" near a copper trace versus a "spurious_copper" were confused. This reflects inherent feature overlap or insufficiently distinct patterns within the training data.

To depict these cases further, Figure 11 presents a selection of misclassified samples. Each sub-image shows a cropped defect region, along with its true class label and the predicted class. Reviewing these misclassifications reveals several potential factors influencing errors, including:

1) Visual Similarity Between Classes: Some defects share common features, making it challenging for the model to distinguish their subtle differences. The learned features may not separate the classes in the feature space. This is shown by overlapping clusters in the PCA visualization (Figure 10).

2) Inconsistent or Noisy Annotations: In several cases, the annotation quality or defect definition may be ambiguous. Slight inconsistencies in labeling can confuse the model during training, and this leads to occasional misclassifications.

3) Limited Representation of Hard Cases: If certain rare defect variations were underrepresented in the training set, the model would have less opportunity to learn the unique patterns required to identify them correctly. Such data imbalances can be addressed by collecting more examples or employing targeted augmentation strategies. None were used in this case, but could be explored in the future for more accurate classification of harder/multiple features.

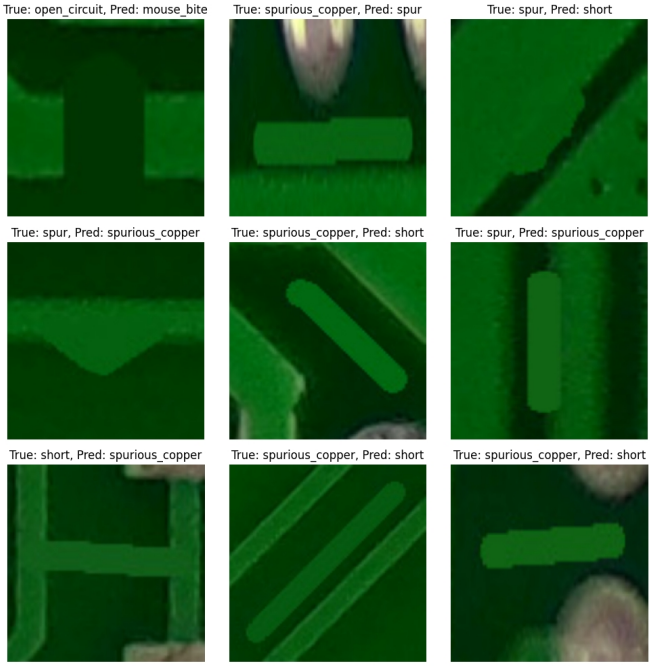

Figure 11: *Examples of Misclassified PCB Defects*

F. Summary of Results

The experiments conducted highlight several findings regarding the efficacy of different PCB defect detection strategies:

1) Superiority of CNN-Based Models: Both the baseline CNN and the deeper CNN models significantly outperformed the classical machine learning methods. The deeper CNN achieved the highest test accuracy at 96%, followed closely by the baseline CNN at 94% accuracy. This performance gap highlights the importance of feature learning directly from raw image data. The CNN models were able to internally represent subtle textural and structural nuances of PCB defects, and this lead to more reliable and generalizable classification results.

2) Effective Handling of Multiple Defect Classes: The confusion matrices, precision-recall curves, and ROC curves showcased that the deeper CNN maintained high precision and recall across all six defect classes. Simpler architectures and classical models could correctly classify the most visually distinct defects, but the CNN-based approaches were better equipped to discriminate between visually similar classes like spur and spurious_copper.

3) Balanced Performance Metrics: The macro-averaged F1-scores and the comprehensive classification reports confirmed that the deeper CNN did more than achieve its high accuracy by excelling at only a

subset of classes. It instead provided balanced, robust performance, minimizing biases toward any particular defect category. This balanced approach is of high importance in real-world scenarios where missing subtle defects could lead to downstream functional failures in PCB products.

4) Limitations of Classical ML Approaches: The SVM, Random Forest, and Logistic Regression models served as valuable baselines, demonstrating respectable performance on certain classes. Notably missing_hole and open_circuit performed the best. Their simpler decision boundaries limited their top-end accuracy, unable to harness the complex image relationships.

5) Insight Through PCA Visualizations and Curves: The PCA visualization provided qualitative insight into why certain classifications proved challenging. It showed that some defect classes formed overlapping clusters in the reduced feature space. The PR and ROC curves offered a threshold-independent view of the models' performance, and also confirming even further the deeper CNN's ability to maintain high precision over a wide range of recall values and its strong ability to separate positive from negative classes.

6) Training Stability and Efficiency: The training history plots revealed that the CNN models converged smoothly and efficiently, reaching peak performance without significant overfitting (minus the one blip in the deeper CNN model). Although the deeper CNN involved more parameters and took relatively longer to train, the incremental gain in accuracy justified the increased complexity. The baseline CNN's rapid convergence and stable performance still made it a viable choice when computational resources or training time are limited (such as in a real time system where YOLO cannot be used).

In summary, the deeper CNN model consistently outperformed all other tested approaches (Table 1) by leveraging its layered convolutional structure to extract more discriminative features. This is evidenced by it's high accuracy, balanced confusion matrices, and favorable PR/ROC curve profiles. The baseline CNN also performed well, verifying that even a simpler architecture can greatly surpass classical ML methods.

IV. Discussion

The goal of this project was to develop an automated vision system capable of detecting and classifying common defects on printed circuit boards (PCBs). This in turn would ultimately lead to improving manufacturing quality control. Early on in the project, several project aims were identified: implementing traditional image processing for defect enhancement, integrating machine learning (ML) for nuanced classification, handling the inherent variability in PCB images, and potentially extending the system to live testing scenarios.

In practice, much of the core functionality envisioned at the proposal stage was successfully implemented. The final code and experiments demonstrated the following accomplishments:

1) Data Preparation and Preprocessing: The project fulfilled the initial milestone of preparing a labeled dataset of PCB defects. Preprocessing techniques were applied to enhance image quality. Although not all forms of advanced adaptive thresholding, template matching, or contour-based methods were deeply explored, the important groundwork of image preprocessing was accomplished. The dataset was effectively parsed, and annotated defects were structured into a unified format conducive to model training and evaluation.

2) Classification Model Using Deep Learning: A primary objective was to leverage machine learning to identify subtle defects that might evade traditional image processing methods. This was successfully achieved through the development and training of convolutional neural network (CNN) models. Both a baseline CNN and a deeper CNN architecture were trained and tested, and they both outperformed classical ML models. This performance indicates that learned feature extraction is well-suited for PCB defect detection. These results aligned with the initial premise that ML would enhance the accuracy and robustness of defect classification.

3) Comparison with Classical ML Approaches: While the original plan suggested integrating classical image processing techniques extensively (e.g., edge detection, template matching) to highlight defects before classification, the final solution shifted focus toward end-to-end deep learning. This shift was primarily chosen due to time constraints, underestimation of task difficulty when using multiple data sets and methods, and what we were learning in class. Some early image enhancement steps were implemented, but the approach ultimately leaned more on CNN-based feature extraction rather than hand-engineered methods. Evaluating scikit-learn models (SVM, Random Forest, Logistic Regression) as baselines demonstrated of the advantages of deep learning models, and ultimately ended with achieving the goal of comparing ML-based solutions with simpler methods.

4) Handling Variability in PCB Images: Variations in lighting, board designs, and defect appearances were addressed indirectly by relying on robust CNN architectures that learn to discriminate subtle differences. With more time or future work this could be explored further. This partial success suggests that even minimal preprocessing and standard CNN architectures can cope with a range of PCB conditions, though additional steps could further improve generalization.

B. Future Work

While the current project has established a solid benchmark, several avenues for future enhancements remain:

1) Data Augmentation and Advanced Preprocessing: Introducing more sophisticated data augmentation techniques

(e.g., slight rotations, brightness adjustments) could improve the model's resilience to environmental variability in PCB imaging. Experimenting with non-linear color transformations or domain adaptation methods might further enhance robustness.

2) Integration of Attention Mechanisms: Future models could incorporate attention layers or transformer-based blocks to help the network focus on critical regions of the PCB. This could improve detection rates for subtle defects and offer more interpretable results by highlighting where the model "looks" when making a decision.

3) Real-Time and Edge Deployment: With deployment as a likely end-goal, optimizing the model for resource-constrained environments (e.g., on-line quality control systems or edge devices in manufacturing lines) could be beneficial.

4) Active Learning and Continual Improvement: As new PCB designs and defect types emerge, implementing an active learning pipeline that allows additional labeled data for challenging classes can keep the model current. Continual or incremental learning approaches could ensure the model remains effective as the defect landscape evolves. This is a rapidly shifting production environment, and with proper work, a robust model could be implemented.

5) Integration with Multi-Task Learning: Looking beyond just classification, future models might be extended to perform localization or severity estimation. This could give manufacturers richer information for targeted interventions. Multi-task learning setups, where the model simultaneously classifies defect type and estimates severity, could streamline quality assurance workflows.

C. Conclusion

The project demonstrated that transitioning from traditional machine learning pipelines to deep learning architectures substantially improved PCB defect detection performance. The methods incorporated thorough data preprocessing, careful comparison between classical ML and CNN-based models, and in-depth performance evaluations using multiple metrics and visualizations. The results validated the hypothesis that CNN models, particularly a deeper variant, will deliver more accurate and generalizable classification outcomes. While the chosen approach met many objectives, the discussion points to multiple strategies for refining and extending the system to meet the evolving demands of PCB manufacturing.

The research has helped clarify the strengths and limitations of various techniques. This provides practical guidance on implementing deep learning methods for future PCB defect detection. As the industry continues to push for higher quality and lower defect rates, approaches that combine robust CNN architectures, adaptive preprocessing, and iterative improvement processes will sought out to achieve these goals.

REFERENCES

[1] *"Convolutional Neural Network (CNN) tutorial," TensorFlow, [Online]. Available: https://www.tensorflow.org/tutorials/images/cnn*

[2] *Muda, Sarjana et al. "PCB Defect Detection, Classification and Localization using Mathematical Morphology and Image Processing Tools." International Journal of Computer Applications 87 (2014): 40-45. DOI:10.5120/15240-3782*

[3] *"Scikit-learn: Machine learning in Python," scikit-learn, [Online]. Available: https://scikit-learn.org/stable/index.html*

[4] *Y. Zhou, M. Yuan, J. Zhang, G. Ding, and S. Qin, "Review of vision-based defect detection research and its perspectives for printed circuit board," Journal of Manufacturing Systems, vol. 70, pp. 557–578, 2023. [Online].DOI: https://doi.org/10.1016/j.jmsy.2023.08.019.*

[5] *Zhang, C., Shi, W., Li, X., Zhang, H. and Liu, H. (2018), Improved bare PCB defect detection approach based on deep feature learning. The Journal of Engineering, 2018: 1415-1420. https://doi.org/10.1049/joe.2018.8275*

[6] *Anoop, K. P., Sarath, N. S., and Sasi Kumar, V. V. (Year), A Review of PCB Defect Detection Using Image Processing. International Journal of Engineering and Innovative Technology (IJEIT), Volume 4. Available: URL.https://citeseerx.ist.psu.edu/documentrepid=rep1&type=pdf&doi=a195ddcb6ea5e25db4d900b4d14b246209974fcf*

[7] Chen, Xing & Wu, Yonglei & He, Xingyou & Wuyi, Ming. (2024). A Comprehensive Review of Deep Learning-based PCB Defect Detection. IEEE Access. PP. 1-1. 10.1109/ACCESS.2023.3339561.