Study of PCB Defect Detection Solutions Using YOLOv8 and a CNN Classifier for Classified Manufacturing Environments

Luis Obed Vega Maisonet

December 4, 2025

Abstract

Classified manufacturing environments cannot rely on commercial automated optical inspection (AOI) systems due to the security risk of exposing sensitive information. A Python framework was implemented to process Printed Circuit Board (PCB) images using a YOLOv8 detector and a CNN classifier on two public datasets: a binarized DeepPCB dataset and a Red, Green, and Blue (RGB) Kaggle PCB dataset. After normalizing both to a standard format, YOLOv8 achieves a mean Average Precision (mAP) at 0.5 equal to 0.983 on DeepPCB and 0.846 on the Kaggle PCB dataset. The CNN achieves approximately 0.99 accuracy on DeepPCB patches but only 0.13 test accuracy when classifying entire Kaggle PCB boards. These results show that defect detection is crucial for complex RGB boards and is preferable to image classification. The study demonstrates that a scalable offline defect-detection system is feasible and could serve as an in-house manufacturing AOI solution.

1. Introduction

PCBs are essential components in modern electronic devices, including smartphones, home appliances, and alarm systems. PCBs undergo rigorous quality control (QC) during manufacturing to prevent the shipment of defective products. Manufacturers use methods such as manual visual inspection (MVI), AOI, and power testing to mitigate defects [1]. However, because of the large number of electronic components and layers in a single PCB, MVI is not feasible. It is also challenging to find quality-control engineers who are proficient in both electronic engineering and system reliability. This challenge is present in relatively simple PCBs and becomes more severe for complex designs that include components requiring software reliability and/or radiofrequency (RF) verification and validation.

Manufacturing companies rely on AOI systems that capture imagery of electronic assemblies and process it through proprietary deep learning models running on industrial servers. These commercial-off-the-shelf (COTS) AOI products are expensive and can pose security risks, since sensitive manufacturing data may be exposed to external suppliers. This is a concern in the defense manufacturing industry, where PCB-based devices are often

developed in classified environments, and sending board imagery outside the secure network may not be acceptable.

As advances in artificial intelligence (AI) and deep learning continue and more pre-trained models are released publicly, the need to rely on a COTS AOI system becomes less compelling, especially for manufacturers with software engineering expertise. This report presents a case study and a practical guide for software engineers working in a classified manufacturing environment to implement PCB defect detection using publicly available deep learning models that can run entirely offline, such as YOLOv8. The case study includes the implementation of the YOLOv8 object detector and a convolutional neural network (CNN) classifier. This case study also illustrates the models' ability to localize and classify six common PCB defect types across two datasets with different imaging conditions.

2. Method

The methodology is based on a framework implementation in Python 3.12. The framework is divided into three main components: a data preparation pipeline that converts public datasets into a consistent format for training and testing; an object detector, YOLOv8, that localizes defects in PCB images; and a CNN classifier that predicts defect categories from images. YOLOv8 and the CNN are implemented using the Ultralytics [2] and PyTorch/Torchvision Python libraries [3], respectively. The two datasets used are DeepPCB and Kaggle PCB [4, 5], which were downloaded from open-source repositories listed in the reference section. Both datasets were used to train and test the designated architectures through a configurable main Python script. The entire architecture is designed to run offline on local hardware [6].

2.1 Data Preparation

The datasets are distributed in their respective directories within the project structure, in raw format, including images and annotations. Both datasets are normalized into a standard structure. The YOLOv8 normalization converts label annotations to YOLO format by obtaining center x, y, width, and height in normalized coordinates (0-1). Lastly, the Ultralytics YOLOv8 library performs spatial resizing and pixel-value normalization internally, rescaling images to a target size of 640 x 640 with padding. The CNN classifier

normalization involves resizing images to 128×128 pixels before training. Afterwards, the training pipeline applies random horizontal flips and small rotations for data augmentation, converts images to tensors in the [0,1] range, and then normalizes each RGB channel using a mean of 0.5 and standard deviation of 0.5, resulting in inputs approximately in the range [-1,1].

Each DeepPCB dataset image is represented in its raw directory structure by three files: an image of the PCB with a defect, an image of the PCB without a defect, and a text file containing bounding boxes and defect class IDs. A data preparation script was developed to scan the test images, match each image to its annotation file, and parse the bounding box coordinates and defect types. The script assigns each image to training, validation, or test split using a fixed 70/15/15 ratio. It uses the same split consistently for both models. The test images for the YOLOv8 detector are saved in split-specific image folders, and the annotations are converted to YOLO-formatted text files with normalized coordinates and zero-based class indices. For the CNN classifier, each annotated bounding box is cropped from the tested image and saved as a small patch into a folder-per-class structure, for example, train/open/ and train/short/.

The Kaggle PCB defects dataset requires a slightly different preparation step. The images are organized by class, and bounding boxes are provided as VOC-style XML files. A second data preparation script scans the image folders, finds the corresponding XML annotations for each image, and normalizes the class names to a shared list of 6 defect categories. As with DeepPCB, the images are split into training, validation, and test sets using the same 70/15/15 split. For the YOLOv8 model, the script writes YOLO-format labels from the XML bounding boxes and copies the original images into the appropriate split folders. For the CNN model, the entire image is used as a training sample and assigned to a class-specific folder based on the primary defect label. This full-image strategy simplifies the data pipeline, but it makes the classification task more challenging on this dataset because many defects occupy only a small region of the image.

2.2 YOLOv8 architecture

The Ultralytics YOLOv8 was used for object detection, initialized from public YOLOv8n weights and then tuned separately on each dataset. The detector takes a PCB image as input

and predicts a set of bounding boxes, each with a confidence score and a defect class label. The YOLO training script is configured with the prepared data.yaml file that points to the train/val/test images and labels produced in the data preparation step. Key hyperparameters include the number of training epochs, the 640 x 640 input image size, and the choice of optimizer and learning rate, which follow the Ultralytics defaults. The YOLOv8 weights and configuration files are stored locally to support offline functionality.

2.3 CNN Classifier architecture

A custom CNN model was implemented, consisting of three convolutional blocks followed by two fully connected layers. Each block applies a 2D convolution, batch normalization, a ReLU activation, and max pooling to progressively increase the number of feature channels while reducing spatial resolution. The final feature maps are flattened and passed through a fully connected layer with ReLU and dropout, followed by a final linear layer that outputs logits over the six defect classes. The network expects RGB inputs of a fixed size of 128×128 pixels. On the DeepPCB dataset, these inputs are the cropped defect patches produced by the preparation script, and on the Kaggle PCB dataset, the inputs are resized into full-board images. The CNN is trained using the cross-entropy loss function and the Adam optimizer, with a configurable learning rate, batch size, and number of epochs.

3. Experiments

To implement reproducible experiments, a single main script controls the training and evaluation by using command-line options for selecting the dataset, choosing the model (YOLOv8, CNN, or both), and adjusting key hyperparameters. By default, the YOLOv8 detector is trained for 50 epochs, with a batch size of 16 and an initial learning rate of 0.01. In comparison, the CNN classifier is trained for 20 epochs, with a learning rate of 0.001 and a batch size of 32. Before each run, a global random seed is set to 42 to improve reproducibility across experiments.

For the CNN architecture, PyTorch DataLoaders are created for the train, validation, and test splits, and standard image transforms are applied. During training, the script logs the training loss, training accuracy, and validation accuracy at each epoch, and saves the model checkpoint that achieves the best validation accuracy. After training, the best model is

evaluated on the test split to report the final classification performance. The best-performing CNN models are saved in the checkpoints_cnn_deeppcb/ and checkpoints_cnn_kaggle/ folders as best_cnn.pth for DeepPCB and Kaggle PCB datasets, respectively.

In the YOLOv8 architecture, the main script calls the Ultralytics training API with the selected dataset configuration and stores all training artifacts, including weights, logs, and plots, in a dedicated folder for each run. After training, YOLO's built-in evaluation function is used to compute mAP at different IoU thresholds and to generate precision–recall curves for each defect class. The YOLOv8 model training artifacts, like weights and learning curves, are stored in runs_yolov8/<run_name>/, and the evaluation artifacts, such as precision–recall curves, confusion matrices, and final mAP metrics, are stored in a separate folder named runs_yolov8/<run_name>_eval/. This common infrastructure allows the same scripts to be reused across datasets while keeping the core logic fully offline and suitable for integration into a classified manufacturing workflow. The script collects key metrics from both architectures, such as the mAP and the classification accuracy, and writes them into a compact JSON summary file, such as results_deeppcb.json and results_kaggle_pcb.json, for later analysis and plotting.

3.1 Dataset

3.1.1 DeepPCB Dataset

The DeepPCB dataset consists of 1,500 image pairs, each containing a reference defect-free PCB image and a tested PCB image with annotated defects. This dataset, in its raw format, has been binarized. The different types of annotation labels are pinhole, open circuit, short circuit, spur, copper, and mouse bite. Figure 1 below shows an example 1 image pair with its text file annotation inside the dataset; the left image is the defect-free image, and the right image is the defective image. Note: Figure 1 borders were added in MS Word to distinguish the image with and without a defect.

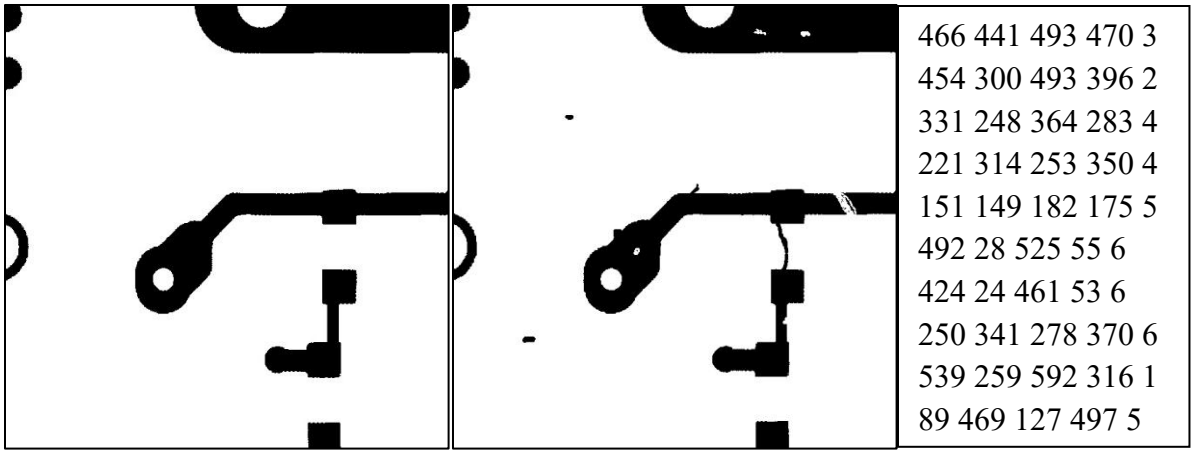

Figure 1. DeepPCB sample 00041000 containing a defect-free image, a defective image, and an annotation file

3.1.2 Kaggle Dataset

The Kaggle PCB defects dataset was provided by the Open Lab on Human Robot Interaction of Peking University. It contains 1386 PCB images organized by defect class, along with VOC-style XML annotation files that specify bounding boxes and labels. This dataset has the same annotation labels as DeepPCB, and its defects are created in Photoshop by Adobe Systems. Figure 2 shows a defect-free image from the Kaggle PCB dataset, labeled sample 01. Figure 3 illustrates the same sample 01 image, with a pinhole/missing hole defect annotation. Lastly, Figure 4 illustrates the XML annotation of the sample 01 image.

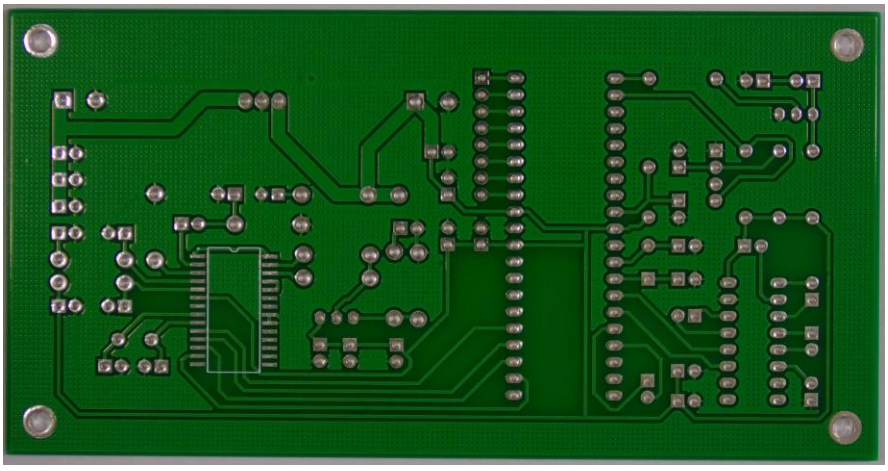

Figure 2. Kaggle PCB sample 01 defect-free image

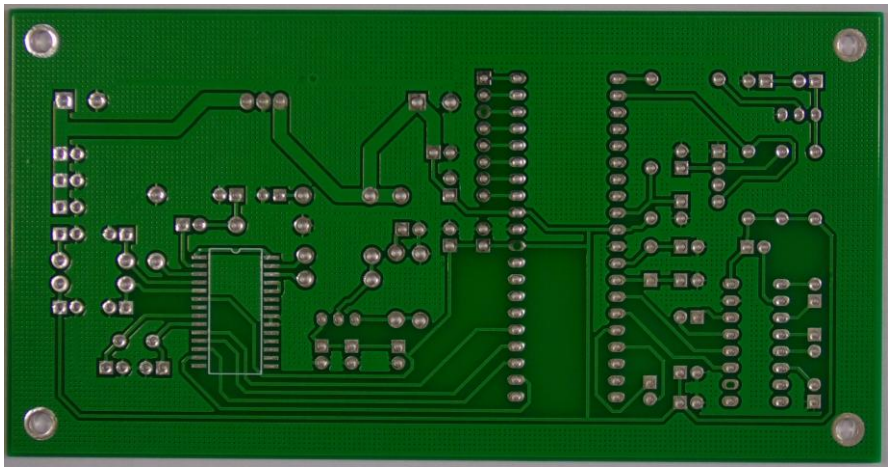

Figure 3. Kaggle PCB sample 01 defective image with a pin hole

```
<annotation>
  <folder>Missing_hole</folder>
  <filename>01_missing_hole_01.jpg</filename>
  <path>/home/weapon/Desktop/PCB_DATASET/Missing_hole/01_missing_hole_01.jpg</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>3034</width>
    <height>1586</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>missing_hole</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>2459</xmin>
      <ymin>1274</ymin>
      <xmax>2530</xmax>
      <ymax>1329</ymax>
    </bndbox>
  </object>
  <object>
    <name>missing_hole</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>1613</xmin>
      <ymin>334</ymin>
      <xmax>1679</xmax>
      <ymax>396</ymax>
    </bndbox>
  </object>
  <object>
    <name>missing_hole</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>1726</xmin>
      <ymin>794</ymin>
      <xmax>1797</xmax>
      <ymax>854</ymax>
    </bndbox>
  </object>
</annotation>
```

Figure 4. Kaggle PCB sample 01 annotation XML file

3.2 Evaluation Metrics

3.2.1 YOLOv8 Evaluation Metrics

The YOLOv8 detector is evaluated using mAP at an IoU threshold of 0.5. This quantity summarizes the trade-off between precision and recall across confidence thresholds. This metric measures how often the predicted bounding boxes overlap with the ground-truth boxes, with an IoU of 0.5 or higher. Additionally, precision over IoU thresholds average values were collected from 0.5 to 0.95 in steps of 0.05. Performing this operation rewards the detector for localizing defects more precisely and penalizes boxes that are correctly classified but poorly aligned.

3.2.2 CNN Evaluation Metrics

Classification accuracy was used as the evaluation metric for the CNN classifier. Specifically, the classification accuracy was used on the validation and test splits. Accuracy is defined as the ratio of correctly predicted samples to the total number of samples in the corresponding split. Here, each image patch is assigned to precisely one of six defect categories. In addition to the scalar accuracy values, the training script stores the loss and accuracy values over epochs. Together, the mAP and accuracy metrics provide a consistent basis for comparing the YOLOv8 detector and the CNN classifier across the DeepPCB and Kaggle PCB datasets.

3.3 Results

3.3.1 YOLOv8 with DeepPCB Results

The YOLOv8 detector achieves a high detection performance on the DeepPCB dataset. The JSON summary for this experiment reports mAP@0.5 = 0.9830 and mAP@0.5-0.95 = 0.7799 on the test split. This indicates that at a standard IoU threshold of 0.5, the detector correctly localizes and classifies almost all defects, even under stricter IoU averaging from 0.5 to 0.95. This high mAP is consistent with the visual appearance of the DeepPCB images, which are binarized and exhibit high contrast between traces, pads, and background. Figure 5 illustrates the precision-recall curve across all dataset annotations.

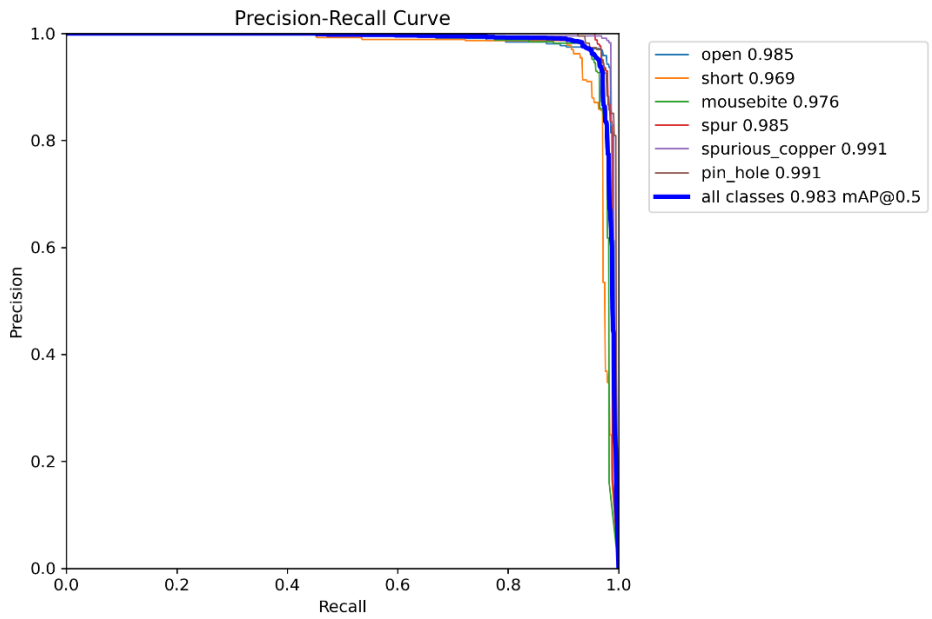

Figure 5. YOLOv8 Detector with DeepPCB Precision-Recall Curve

The precision–recall behavior illustrated in Figure 5 is characteristic of a strong model, with high precision maintained over a wide range of recall levels. All classes achieve per-class AP at IoU 0.5, ranging from 0.96 to 0.98. A confusion matrix is a table that compares the ground truth class labels on one axis with the predicted class labels on the other. In other words, each cell (i, j) shows how many examples of class i were predicted as class j. The large values on the diagonal indicate correct predictions. The off-diagonal values correspond to misclassifications between pairs of classes. Figure 6 below illustrates the YOLOv8 detector confusion matrix for the DeepPCB dataset. Here, the entries are normalized so that each row sums to 1, making it easier to see where the model tends to confuse specific defect types regardless of class frequency.

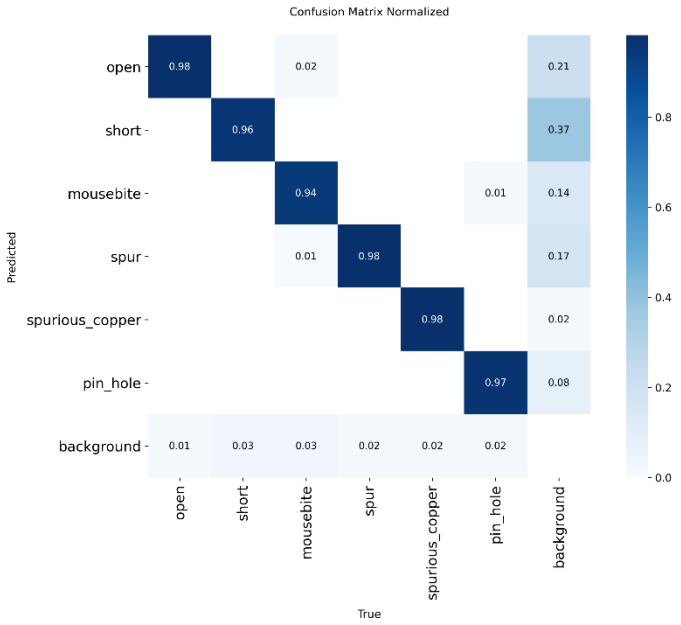

Figure 6. YOLOv8 Detector with DeepPCB Confusion Matrix

The Figure 6 matrix shows that the detector rarely confuses one defect type with another. For each of the six classes, most ground-truth instances are correctly assigned to the corresponding predicted class, with only a small number of off-diagonal errors. Figure 7 shows qualitative examples of YOLOv8 predictions on DeepPCB samples. Each panel corresponds to a different tested PCB image, and the predicted bounding boxes are overlaid with color-coded labels for the six defect types. It demonstrates that the model can simultaneously detect multiple defects on the same board. Most predicted bounding boxes cover the defective regions, and the labels agree with the ground truth. This visually confirms the strong diagonal structure observed in the confusion matrix and the high mAP values reported for this dataset.

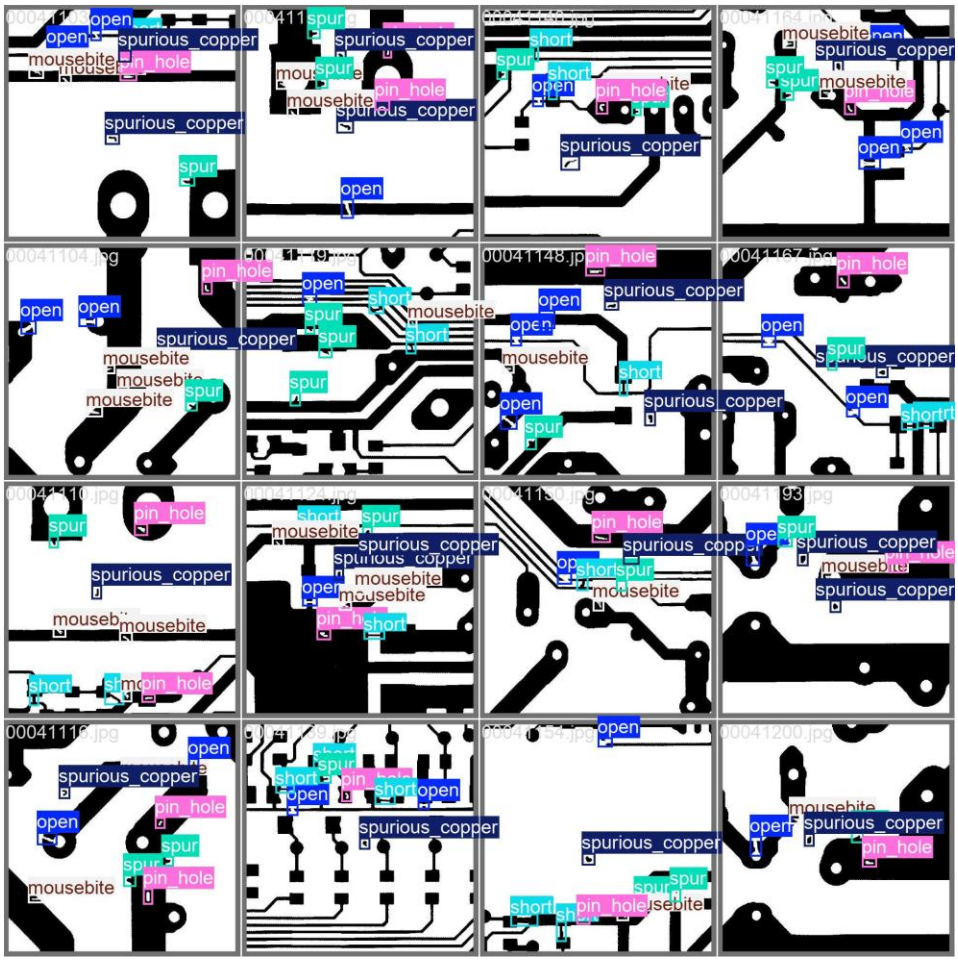

Figure 7. YOLOv8 Detector with DeepPCB Predictions

3.3.2 YOLOv8 with Kaggle PCB Results

The YOLOv8 detector performs well on the Kaggle PCB dataset, but its performance drops compared to DeepPCB. The JSON summary file shows that mAP@0.5 = 0.8464, and mAP@0.5-0.95 = 0.4245 on the test split. The detector can localize most defects at the 0.5 IoU threshold, but the lower mAP range indicates that precise bounding-box localization is more challenging in this dataset. This result is logical since the Kaggle images are full-color boards with more complex backgrounds and varying lighting conditions. This means the YOLOv8 detector can localize defect regions but may occasionally produce bounding boxes that are slightly misaligned when a stricter IoU is applied. Figure 8 illustrates the precision-recall curve across all dataset annotations.

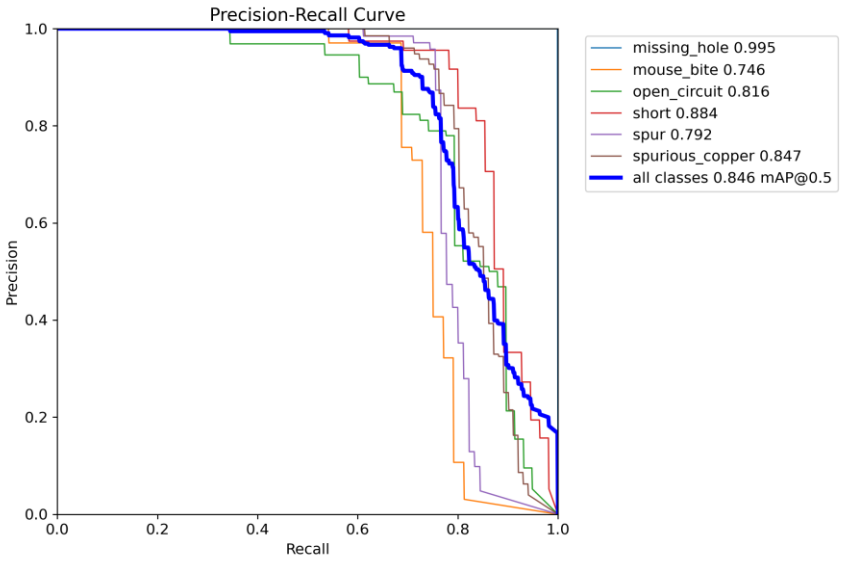

Figure 8. YOLOv8 Detector with Kaggle PCB Precision-Recall Curve

The Kaggle PCB curves shown above illustrate a greater trade-off between precision and recall across classes. The missing hole class achieves the highest area under the curve (mAP@0.5 = 0.99), whereas classes such as mouse bite, spur, and spurious copper have lower areas under the curve. This indicates a greater quantity of false positives and missed detections at higher recall levels. The thick blue curve peaks at mAP@0.5 = 0.85 and is consistent with the earlier mAP values. The plot confirms that YOLOv8 remains effective on the Kaggle PCB dataset, but detecting defect types reliably is more challenging in RGB images than in binarized DeepPCB images. Figure 9 illustrates the confusion matrix for the YOLOv8 detector on the Kaggle PCB dataset.

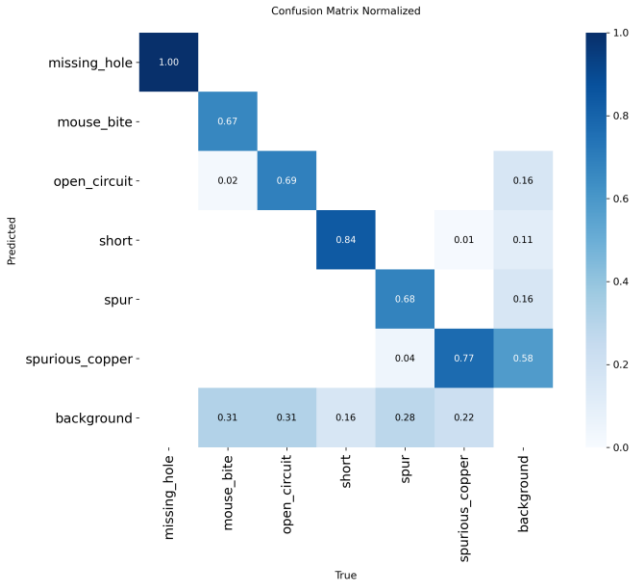

Figure 9. YOLOv8 Detector with Kaggle PCB Confusion Matrix

The confusion matrix for the Kaggle PCB dataset has a stronger diagonal concentration but is noticeably less “clean” than the DeepPCB matrix. There are more cells with non-zero values off the diagonal, indicating that the detector confuses some defect types under imaging conditions. Classes such as mouse bites or spurious copper are often misclassified as other defect types. This pattern matches the lower mAP values at 0.5-0.95 reported for Kaggle PCB. Figure 10 shows qualitative examples of YOLOv8 predictions on Kaggle PCB test images.

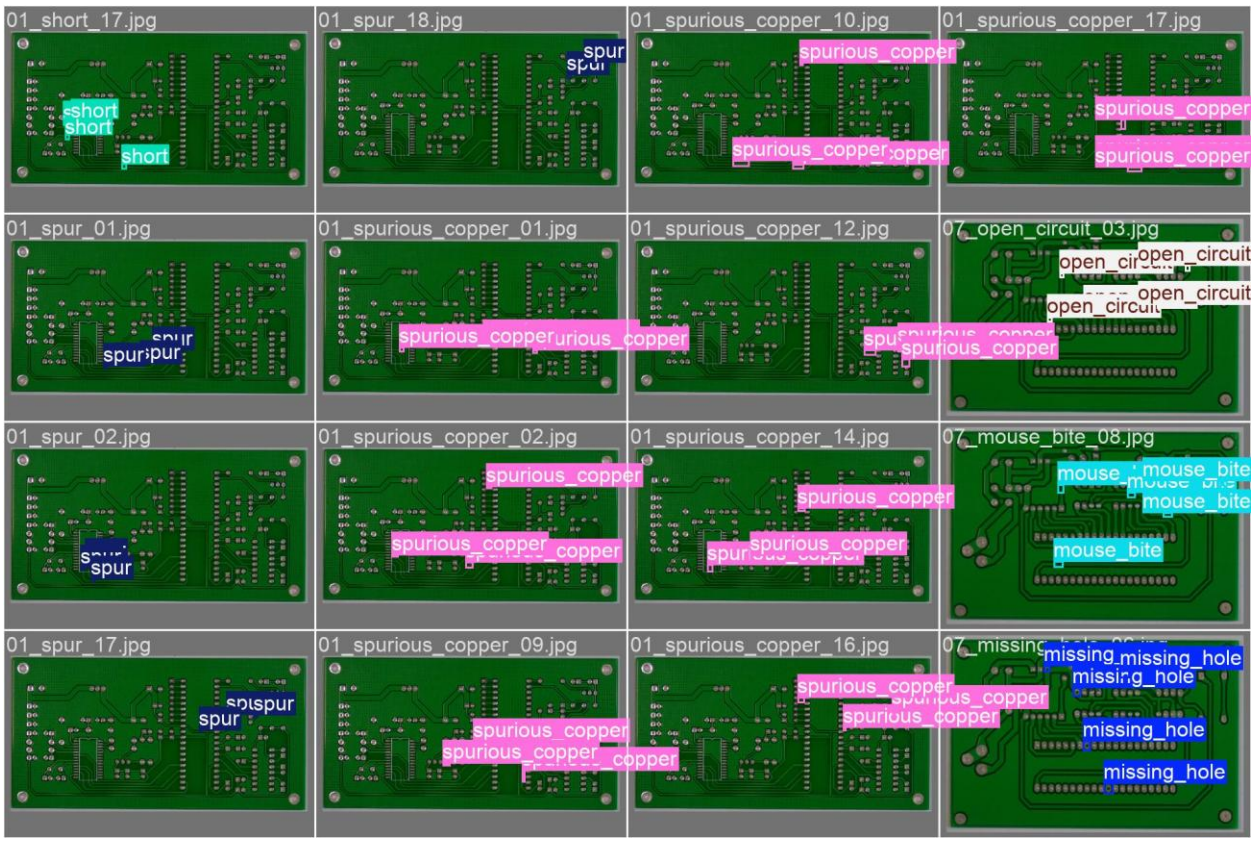

Figure 10. YOLOv8 Detector with Kaggle PCB Predictions

Each panel of Figure 10 corresponds to an RGB image of a colored PCB. Every image contains the predicted bounding boxes overlaid with color-coded labels for the six defect types. Compared to the DeepPCB images, the boards contain richer backgrounds and non-uniform lighting. It illustrates that the detector can still localize and classify multiple defects per board. It highlights the challenges observed in the mAP and confusion-matrix results. Finally, this highlights that RGB PCB images make detection and classification tasks significantly more challenging than in a binarized DeepPCB setting.

3.3.3 CNN with DeepPCB Results

The CNN classifier performs well when trained on cropped defect image patches. The best model was selected based on the validation accuracy. The validation accuracy is 0.9950, and the test accuracy is 0.9938. Both values indicate that the network rarely misclassifies a patch as belonging to any of the six defect categories. The training and validation curves, shown in Figure 11, stabilize quickly and remain close to each other, suggesting that the model is not overfitting despite its high performance. This behavior is consistent with the

structure of the DeepPCB patches: after localization, each patch contains a clear, high-contrast defect pattern against a relatively simple background, which facilitates the learning of features by a CNN classifier. In Figure 11, the training curves increase rapidly and converge near 0.99 within the first 10–12 epochs, with a gap between the training and validation accuracies.

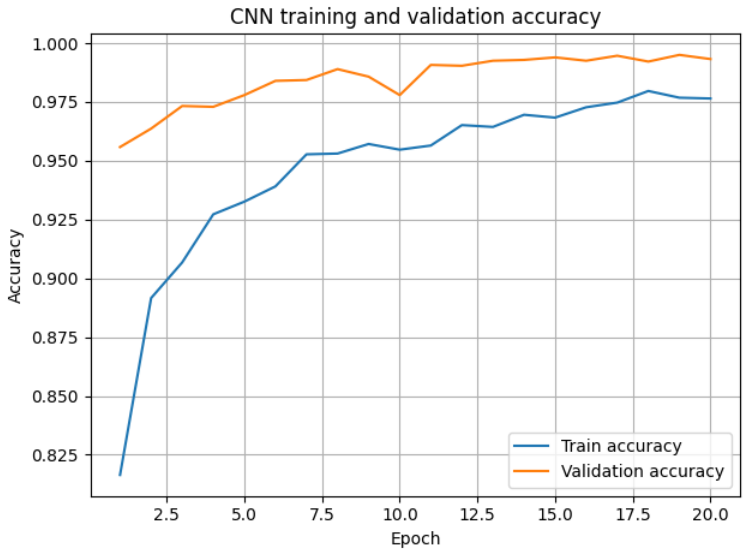

Figure 11. CNN training and validation accuracy on the DeepPCB dataset

Figure 12 shows the CNN training loss curve. The training loss decreases smoothly from about 0.7 to below 0.1. These results confirm that this type of CNN learns the defect classes quickly and generalizes well, which is consistent with the reported validation accuracy.

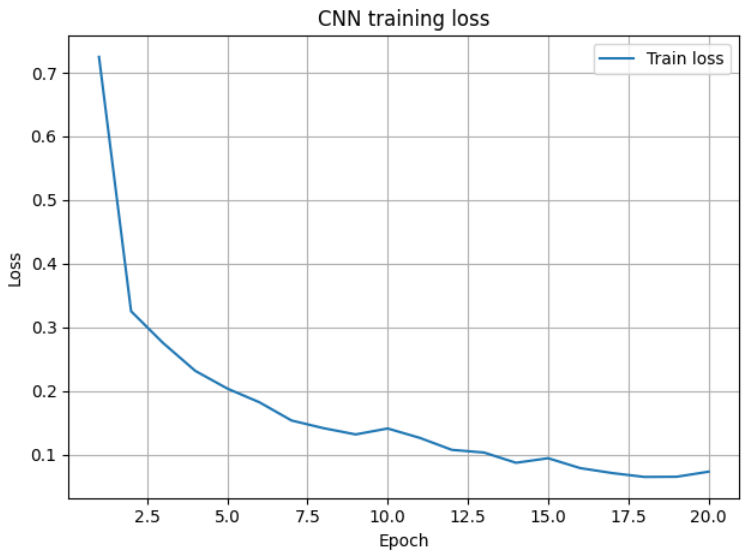

Figure 12. CNN training loss on the DeepPCB dataset

3.3.4 CNN with Kaggle PCB Results

On the Kaggle PCB dataset, CNN struggles when trained directly on full-board RGB images. Using the same architecture and training setup, the best model achieves validation accuracy of 0.2427 and test accuracy of 0.1333, which is close to the random-guess baseline of $1/6 \approx 0.167$ for six classes. Despite running for 20 epochs with data augmentation and standard optimization settings, the network fails to learn features that generalize beyond the training split. This poor performance reflects the difficulty of the task formulation: each input image contains a complete PCB with minor, localized defects embedded in a large amount of background circuitry. Since the CNN classifies a single label with no additional information about defect localization, it must perform both localization and classification, which is much more complex than the patch-based setup used with DeepPCB. The training and validation curves are shown in Figure 13.

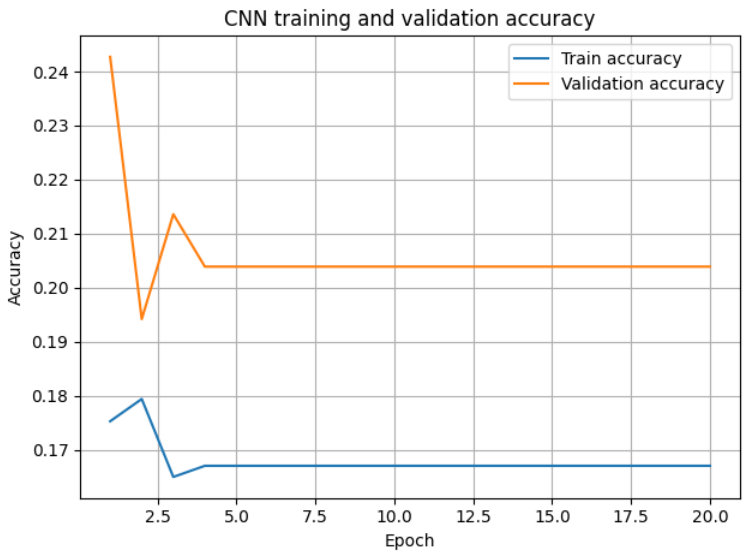

Figure 13. CNN training and validation accuracy on the Kaggle PCB dataset.

Figure 13 shows that the training and validation accuracy remains low, ranging from 0.20 to 0.24. The training accuracy stays close to 0.17. This value exceeds the random-guess baseline of 1/6 (approximately 0.167) for six classes. The training loss curve shown in Figure 14 drops sharply during the first epoch and then remains around 1.7. These patterns confirm that the CNN fails to learn features when trained on full PCB RGB images and behaves like a near-random classifier on the Kaggle PCB dataset. These results are in line with the reported accuracy values.

3.4 Analysis and discussions

The experiments demonstrate that the dataset characteristics and the problem formulation influence the effectiveness of deep learning models for PCB defect detection. The DeepPCB dataset, with both the YOLOv8 detector and the CNN classifier, achieves near-saturation performance. YOLOv8 achieves mAP@0.5 = 0.9830 with substantial precision–recall curves and a nearly diagonal confusion matrix, while the CNN attains validation and test accuracies above 0.99 when trained on localized patches.

These results indicate that once defects are presented as high-contrast binary patterns, either as full images for detection or as cropped patches for classification, the visual distinction between defect categories is clear enough that relatively compact models can learn robust features without severe overfitting. The Kaggle PCB dataset shows the

limitations of treating PCB inspection as a simple whole-image classification problem. YOLOv8 still performs reasonably well, with a mAP@0.5 = 0.8464. The mAP@0.5-0.95 = 0.4245 indicates noisier precision–recall curves, indicating that precise localization and class separation are more difficult on full-color boards. When trained on resized full-board images, the CNN's validation accuracy saturates near 0.24, and its test accuracy falls to 0.1333, only slightly above random guessing for six classes. The flat learning curves and the constant loss confirm that the network fails to discover patterns. This suggests that, for realistic RGB PCB imagery, an image classifier alone is insufficient and that explicit defect detection is crucial.

Comparing the two architectures suggests a practical design pattern for PCB inspection systems. YOLOv8, trained in a fully supervised detection, is better suited across datasets and remains effective even when the background is cluttered. The CNN performs best when given prelocalized patches, as in the DeepPCB architecture. These results point toward a two-stage workflow in which an object detector, such as YOLOv8, first proposes candidate defect regions, and a downstream classifier refines the defect type if necessary. In an industrial deployment, this separation of responsibilities would allow the detector to focus on finding “anything suspicious” on a board. At the same time, specialized classifiers could be tuned for defect families or board designs.

This study demonstrates that high-quality PCB defect detection is achievable using only publicly available models and datasets in a fully offline environment. All training, evaluation, and plotting are performed locally, and model weights, configuration files, and metrics are stored in project-specific directories rather than in external services. This is particularly important for classified manufacturing environments, where board imagery and defect annotations cannot leave the secure network. The results on DeepPCB demonstrate that such an offline setup can match or exceed the performance typically reported in the literature. In contrast, the Kaggle results reveal the challenges that remain when moving toward more realistic scenarios. These insights provide a foundation for future work to improve localization accuracy, handle more complex boards, and integrate additional data sources, such as solder paste inspection or X-ray imagery, within the same offline framework.

4. Conclusion

This report investigated whether modern deep learning PCB defect detection can be implemented in a classified manufacturing environment without depending on AOI systems. Starting from public datasets and open-source tools, an offline framework was implemented to prepare PCB imagery. This framework trains a YOLOv8 object detector and evaluates a CNN classifier across two datasets with different imaging characteristics. The experiments showed that on the binarized DeepPCB dataset, both models achieve near-saturation performance. In contrast, on the more realistic, full-color Kaggle PCB dataset, YOLOv8 still provides valid detections, but the CNN struggles to classify whole-board images. These results confirm that deep learning can match or exceed commercial AOI performance on well-structured data and remain viable under more challenging conditions when configured as a detector-first pipeline.

From the perspective of AOI and quality inspection, the study suggests a practical way for manufacturers to reduce dependence on expensive commercial-off-the-shelf (COTS) solutions. A lightweight YOLOv8 model, trained and executed entirely offline, can act as the core of an in-house AOI system that flags suspicious regions on each board. A downstream CNN or rule-based module can then refine these detections or integrate them with existing electrical and functional tests. Because the framework uses standardized data formats, configurable YAML files, and a single main script to control experiments, it is inherently scalable: new PCB designs, additional defect types, or entirely new datasets can be incorporated by extending the preparation scripts and configuration files rather than re-architecting the system. In a production setting, this design could be integrated into existing QC workflows as an additional inspection stage or as a software layer on top of traditional AOI hardware.

At the same time, the work highlights several limitations and practical barriers to adopting these techniques in classified manufacturing workplaces. Many secure facilities lack dedicated GPU resources, or management may be reluctant to invest in hardware and personnel for deep learning, especially when existing AOI equipment is already in place. Training and maintaining models require software engineering and machine learning expertise that may not be available in all manufacturing teams, and labeled PCB defect data

for proprietary designs is not always easy to obtain. Furthermore, the experiments in this report are limited to two public datasets and single-image inspections; they do not yet cover multi-layer boards, multi-camera AOI setups, or the full range of environmental variation encountered on a real production line.

Despite these constraints, the case study demonstrates that, once a modest level of computing and expertise is available, implementing an offline, secure deep learning pipeline for PCB defect detection is technically feasible. A single workstation with a commodity GPU is sufficient to train and deploy the models used in this project, and the open-source nature of the framework allows organizations to adapt and extend it without licensing restrictions. For classified manufacturers willing to make this investment, the approach outlined here offers a path toward scalable, customizable AOI that keeps sensitive imagery on-premises while leveraging state-of-the-art vision models. Future work can build on this foundation by incorporating additional modalities, exploring semi-supervised learning to reduce labeling effort, and tightening the integration between deep learning inspection and broader quality management systems.

5. References

[1] Andwin Circuits, “Quality control in PCB manufacturing: Ensuring design specifications are met.” [Online]. Available: https://www.andwinpcb.com/quality-control-in-pcb-manufacturing-ensuring-design-specifications-are-met/

[2] Ultralytics, “YOLOv8,” *Ultralytics YOLO Docs*. [Online]. Available: https://docs.ultralytics.com/models/yolov8/

[3] J. E. Solem, *Programming Computer Vision with Python*. Pre-production draft, 2012.

[4] S. Tang, F. He, X. Huang, and J. Yang, “DeepPCB: A dataset for PCB defect detection,” GitHub repository, 2019. [Online]. Available: https://github.com/tangsanli5201/DeepPCB/tree/master/PCBData

[5] A. Akhatova, “PCB Defects,” Kaggle dataset, 2019. [Online]. Available: https://www.kaggle.com/datasets/akhatova/pcb-defects

[6] Ultralytics, “Configuration,” *Ultralytics YOLO Docs*. [Online]. Available: https://docs.ultralytics.com/usage/cfg/