DataScientest

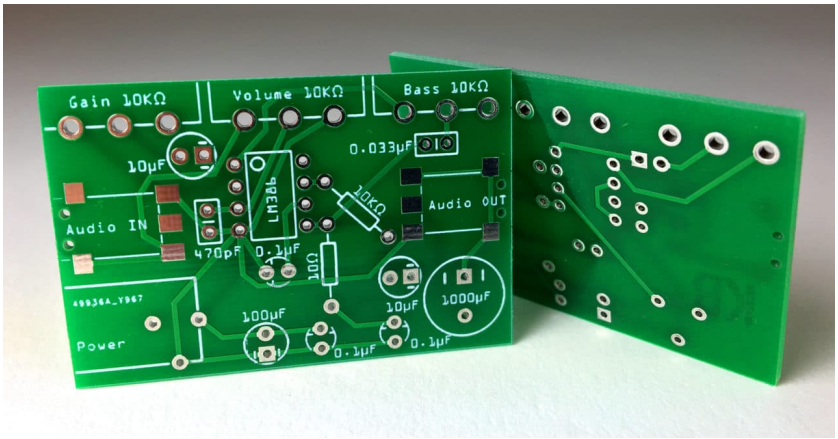
Detection and Classification of Defects on Printed Circuit Boards with Machine Learning

Faiza Waheed
w.faiza@gmx.de
https://github.com/wfaiza

Niels Hartano
niels@gmx.de
https://github.com/taubenus

Gernot Gellwitz
gernot@gmx.de
https://github.com/Kathartikon

Supervisor: Alban Thuet
alban@datascientest.com
https://github.com/lokilone



https://github.com/wfaiza/PCB_Defects_Detection

May 27, 2024

Contents

Abstract

This report presents a machine learning approach for detecting and classifying defects on printed circuit boards (PCBs). The aim is to enhance the quality control process in PCB manufacturing by leveraging advanced computer vision techniques in observing and identifying various defects.

Introduction

Printed Circuit Boards (PCB's) are essential components in nearly all electronic devices. Ensuring their quality is critical, as defects can lead to device malfunctions or failures. Visual inspection, defect detection and recall are some of the most complex and expensive tasks for PCB manufacturing companies. Over the years, Printed Circuit Boards have become much smaller and more densely packed with components making the scalability of visual inspection harder. Traditional inspection methods, often manual, are time-consuming and prone to human error.

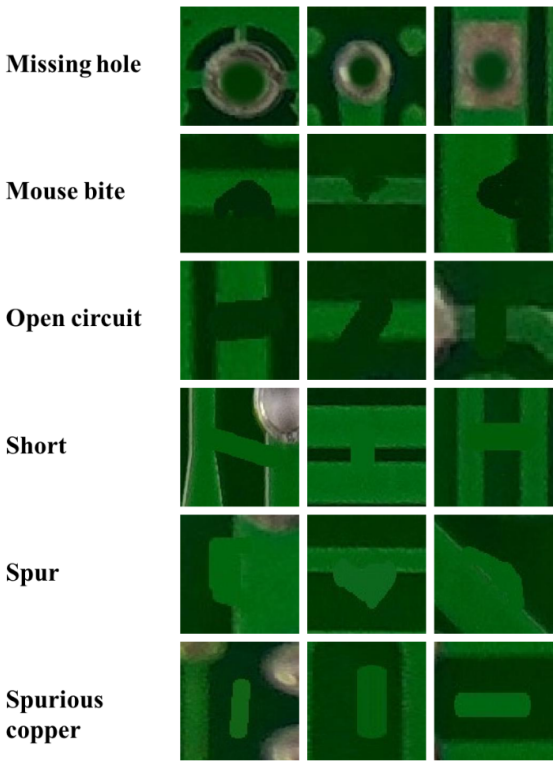


Figure 1: Sample defects explored in this project

Machine learning, particularly deep learning, has shown significant promise in automating and improving the accuracy of defect detection and classification in PCB's. By training models on annotated images of PCB's, these systems can learn to identify various types of defects such as solder joint issues, component misalignment, and

surface contamination (See Fig. 1.). This quality assurance procedure ensures that the product quality is validated before it is marketed.

This project focuses on two approaches for defect detection and classification: the first approach is based on the VGG16 network, the second is based on a manually designed Convolution Neural Network U-Net model. The former makes use of so-called bounding boxes to detect defects, while the latter uses mask segmentation for detection and classification of defects.

VGG16 can be downloaded as a pre-trained model from the TensorFlow Keras library. Whereas we developed and implemented the U-Net model from scratch. VGG16 therefore can be utilized for transfer learning.

A public PCB dataset containing over 10,000 images with 6 kinds of defects (Missing hole; Mouse bite; Open circuit; Short circuit; Spurious copper; Spur) was used for detection, classification and reporting tasks. This dataset is provided for public use and hosted on Kaggle.com, which is a community for data scientists and ML developers. The dataset is located at:
https://www.kaggle.com/datasets/akhatova/pcb-defects.

The dataset hosted on Kaggle is effectively sourced from the Open Lab on Human Robot Interaction of Peking University from
https://robotics.pkusz.edu.cn/resources/datasetENG/.
This is a large dataset of more than 10,000 PCB images with a total of around 22,000 annotated defects (classification and bounding box for defects), which we used to train and evaluate our models. The goal was to develop a robust system capable of detecting and classifying defects with high accuracy and efficiency.

Some images only contained one defect, while others contained multiple defects (See Fig. 2.), but the class of defects for a single image was the same.

Over all, 6 different defects were considered for this project:

- Mouse Bites
- Missing Holes
- Short circuit
- Spurious Copper
- Spur on trace
- Open Circuits

Initially the database seemed balanced, but that was from observing the class labeling for individual images. The defects appeared to be fairly distributed in the

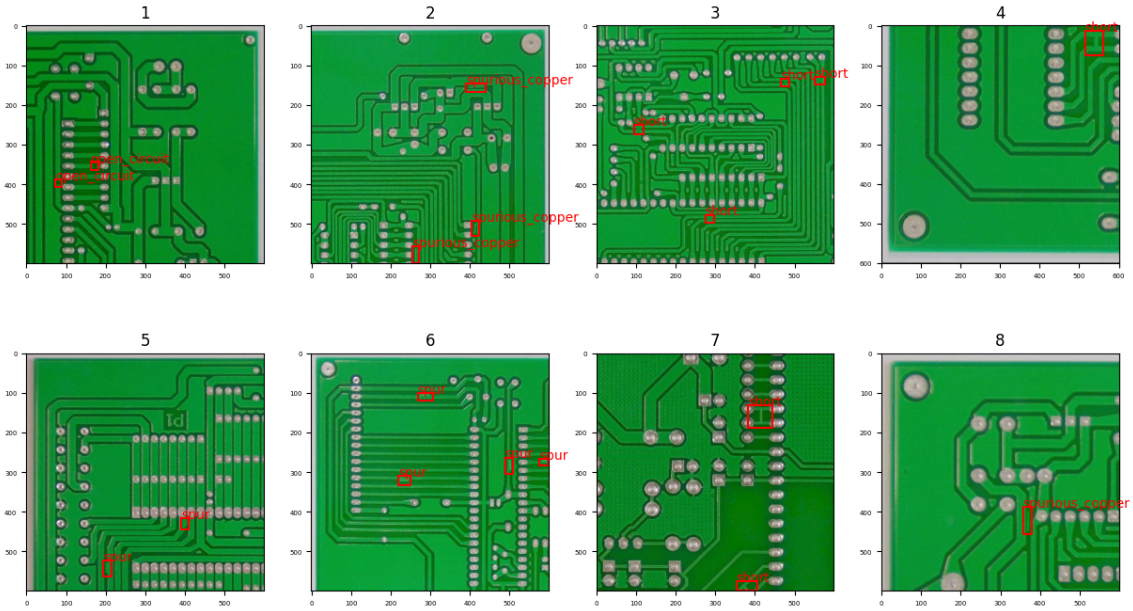


Figure 2: Sample images from the PCB dataset with annotated defects

dataset, with the most common defect being "Mouse Bites" and the least common defect being "Shorts" (See Fig. 4.).

The main objectives were to:

- Create a data-frame from around 10.000 ".xml" annotation files in a ".csv" format containing the dimensions of the bounding boxes, size of the pictures and the class of defect.
- Pre-process the data which included resizing, sorting, ensuring that feature components were not distorted.
- Populating the dataset by implementing image augmentations.
- Training machine learning model to detect and classify defects.
- Evaluate the model's performance and optimize it for deployment in a production environment.

1.1 Pre-processing

A high-quality dataset is crucial for training an effective model. Data augmentation is a crucial technique in machine learning, particularly for tasks involving image

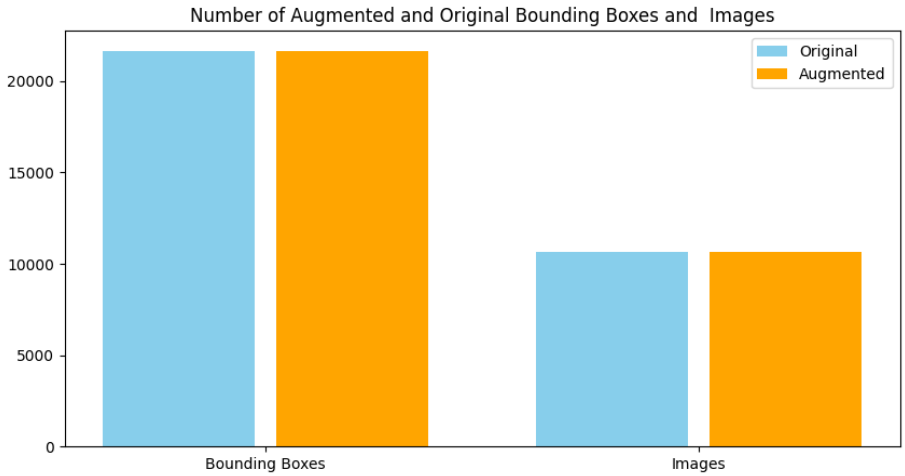


Figure 3: Ratio of Number of defects to Number of images for VGG16 model(revisit this image)

data, such as object detection and classification of defects on printed circuit boards (PCB's). By artificially expanding the training dataset through transformations like rotations, flips, scaling, and translations, data augmentation helps improve the robustness and generalization ability of the model. This process mitigates over-fitting by exposing the model to a diverse set of variations and scenarios that it might encounter in real-world applications. Consequently, data augmentation enhances the model's ability to accurately detect and classify defects, even when faced with new or slightly altered images, thereby improving its overall performance and reliability in practical deployment. Two approaches were considered for data augmentation: Using a library or implementing the augmentations by hand. The former is more convenient and less error-prone, while the latter offers more flexibility and control over the augmentation process.

1.1.1 Augmentations via Albumentations

The library-based approach made use of the library "Albumentations", specifically techniques available like random brightness or contrast, random cropping of the image, rotation, horizontal or vertical flipping, including random sun flares or changing of the hue saturation value (See Fig. 5.).

Problems that occurred during this process were that the cropping of the images may lead to the loss of the defect, if the defect is located outside of the cropped part of the image, which is not desired. In this case the defect would not be detected

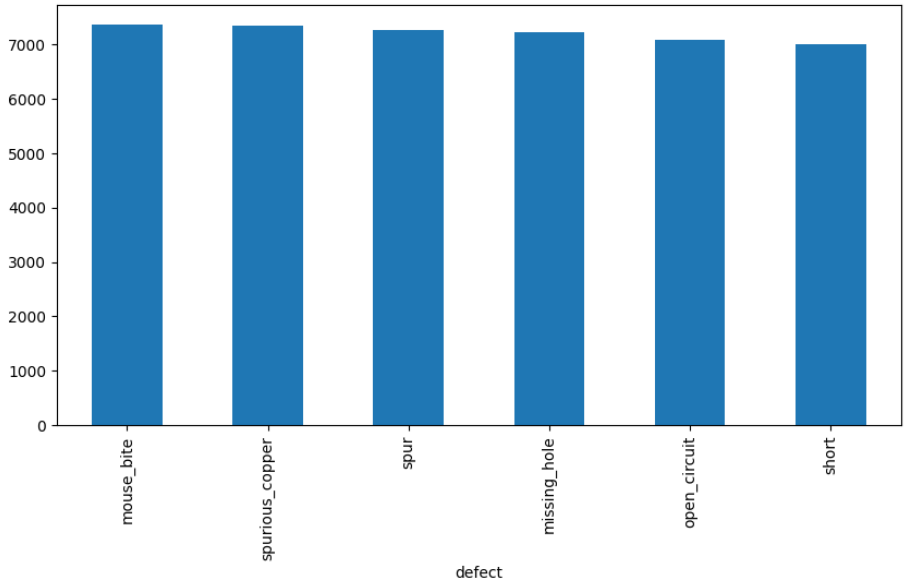


Figure 4: Defect distribution

by the model. Another problem was that in the case of rotation the defects were not correctly placed as can be seen in Fig. 5. This observation is why cropping and rotation were not used in the final VGG16 model.

1.1.2 Manual Augmentations

Let us now discuss the pre-processing that went into preparing the dataset before and after augmentation so that it would not lead to insufficient or inefficient training.

Early on after observing the size of images (dim:600x600x3), we decided to crop the images to easily processable dimensions i.e. 100x100. Another decision was to convert the images to gray scale since that would improve computing power immensely while there would be no observable loss in the features of the images.

We observed various issues during the process of implementing on-the-fly augmentation by data generators like ImageDataGenerator on the dataset. We managed to solve some of those like instances where masks and labels were incorrectly referenced to the original image or adaption to our multi output structure (segmentation and classification) or insufficient control over the type of augmentation. Others were harder to come by because they were intrinsic and needed solutions too complex to be practical. Like that - similarily to Albumentations - the augmentation provided by DataImageGenerator performed rotation and zoom transformations differently on

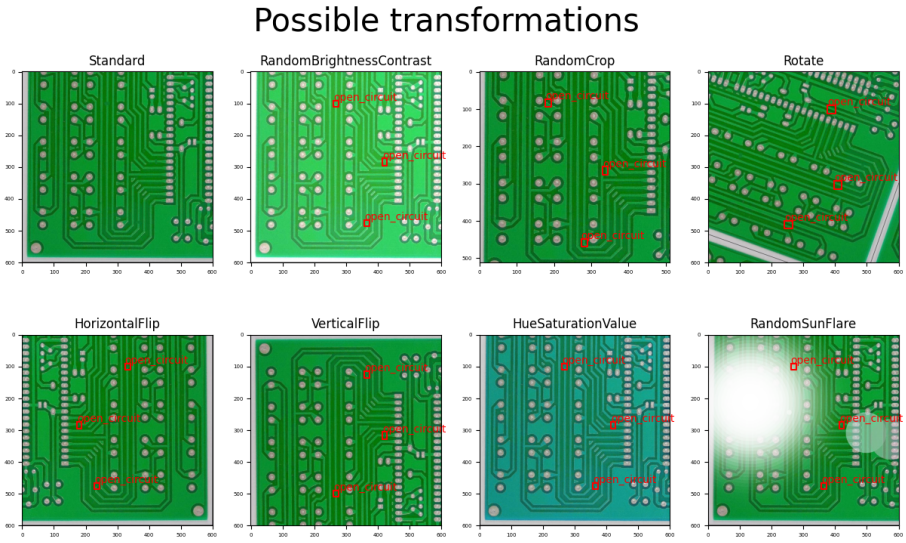


Figure 5: Possible augmentations by "Albumentations".

the images and their corresponding masks so that images and masks would not be aligned anymore after augmentation, or defects that were wholly or partially cropped by shift transformations. Considering all this we concluded that it would be beneficial to invest time in implementing manual augmentation for the cost of having to save all the augmented dataset to disk. For training a robust and reliable U-Net model, the following augmentation techniques were implemented (See Fig. 8.).:

- Rotation the image
- Shifting the image horizontally
- Shifting the image vertically
- Shearing the image
- Enlarging or reducing the image
- Horizontally flipping the image
- Introducing Gaussian noise to the image

To implement cropping on the dataset, we had to keep in mind that the relevant label/defect class would be properly referenced. In our case, we had to introduce the "none" class to offset the generation of cropped images that had no "defects".

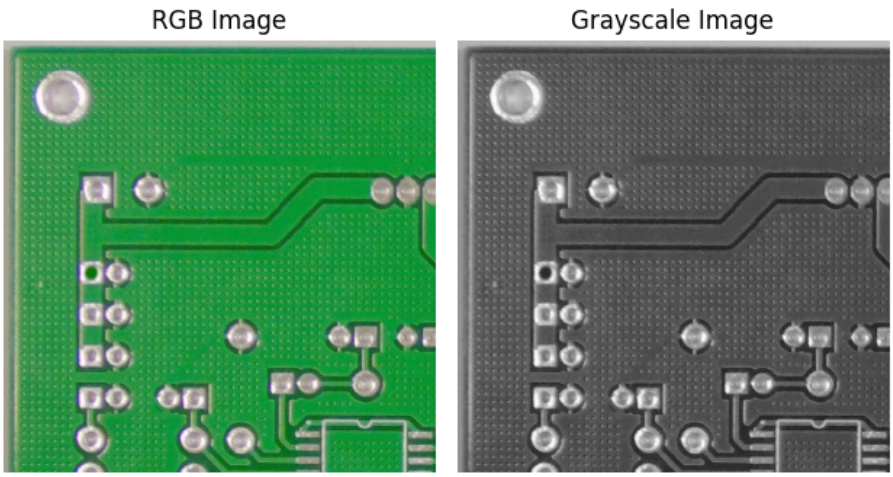


Figure 6: Colored vs. Grayscale image

While preprocessing the dataset, we also observed that sometimes the defect would be located in the cropping boundary of the image. If we implemented the cropping without any intervention, the cropped image would not provide a good feature set for the model to train on. Hence we implemented a boundary shifting function which ensured that the defect would not be cropped.

Another observation was that while augmentation, if the defect was located at the boundary of the image, sometimes it would remove the defect from the image after implementing the augmentation technique. Hence we had to implement a check to ensure that the relevant features were not lost. With these checks and balances, we managed to ensure that the dataset for training the model was balanced, relevant and not suffering from feature loss. As already mentioned, the images in the dataset have single and multiple defects in a single instance (image). Therefore, we had to cater for the cases in which there were multiple defects in a single image so that the model could process the mask and label for the defects accordingly. We brainstormed on how to handle such instances and agreed on removing multiple defects from a single image and separating all defects into single image instances. For this we implemented a defect separation function.

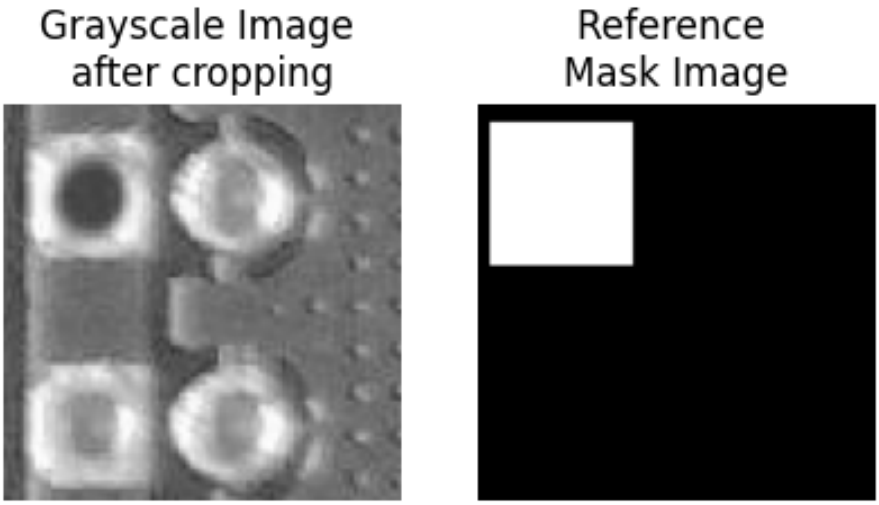


Figure 7: Cropping image and mask to 100x100 dimension

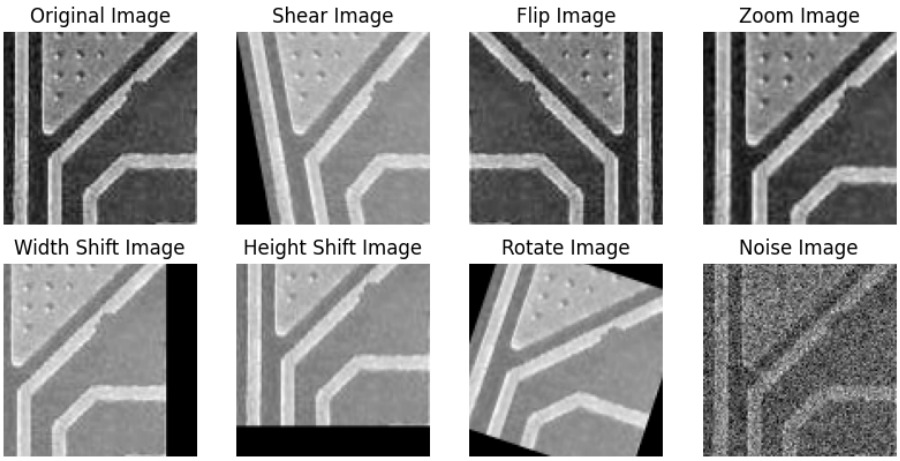


Figure 8: Manually implemented augmentations

Modelling

The methodology of this project involves several key steps: data frame generation and preprocessing, model design and training, and evaluation.

2.1 Model Design and Training

2.1.1 VGG16

The VGG16 model is a convolutional neural network architecture that has been widely used for image classification tasks. It consists of 16 layers, including 13 convolutional layers and 3 fully connected layers. The model has been pre-trained on the ImageNet dataset, which contains millions of images across thousands of classes. By leveraging transfer learning, we can take advantage of the features learned by the model on ImageNet and fine-tune it on our PCB dataset. The model was implemented using the TensorFlow and Keras libraries.

VGG16 requires the following additional preprocessing steps:

- Resizing images to a 224 by 224 image size to fit the model input requirements. All colors where kept
- Normalizing pixel values to the range [0, 1].

Furthermore the dimensions of the bounding box where normalized and centered. The defects had to be split up by using a One Hot Encoding. The dataset was then split into a training and a validation set. TensorFlow additionally requires the transformation of the dataset into a tf.data.Dataset object.

Two approaches were tested: freezing (keeping) all pretrained layers and only adding a flatten layer and an detection and an classification head and unfreezing all layers.

Using two different heads enables the user to use different loss functions and metrics for the detection and classification task. For the detection task the loss function GIoU and metrics One-Hot-IOU was used, while for the classification task the loss function categorical cross-entropy and metrics accuracy were used.

To avoid unnecessary computation, callbacks were implemented such as early stopping and model checkpoint.

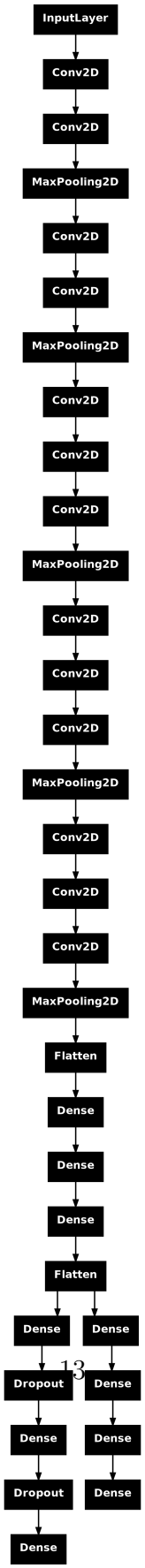


Figure 9: Vgg16 with two heads.

2.1.2 U-Net

For the development and implementation of our machine learning model, we went through many design iterations to finally decide on the RES-UNET model scheme (See Fig. 10.).

The RES-UNET model provides the combined qualities of a Residual network connection to enhance feature extraction at every stage of the Unet network architecture. With the help of having the Residual connection, the model has ease of learning by removing the vanishing gradient problem along with robust feature extraction. The Unet architecture ensures with the help of skip connection, that the high resolution features, which we need in our case for the defect segmentation task (object detection), are combined from the encoder and decoder to preserve spatial information.

As our task is to classify and detect the defects, we need both segmentation and classification outputs. This model handles both required outputs for a single instance simultaneously. We did entertain the idea of having separate models for obtain the two outputs, but decided to explore this combinatorial model architecture implementation instead.

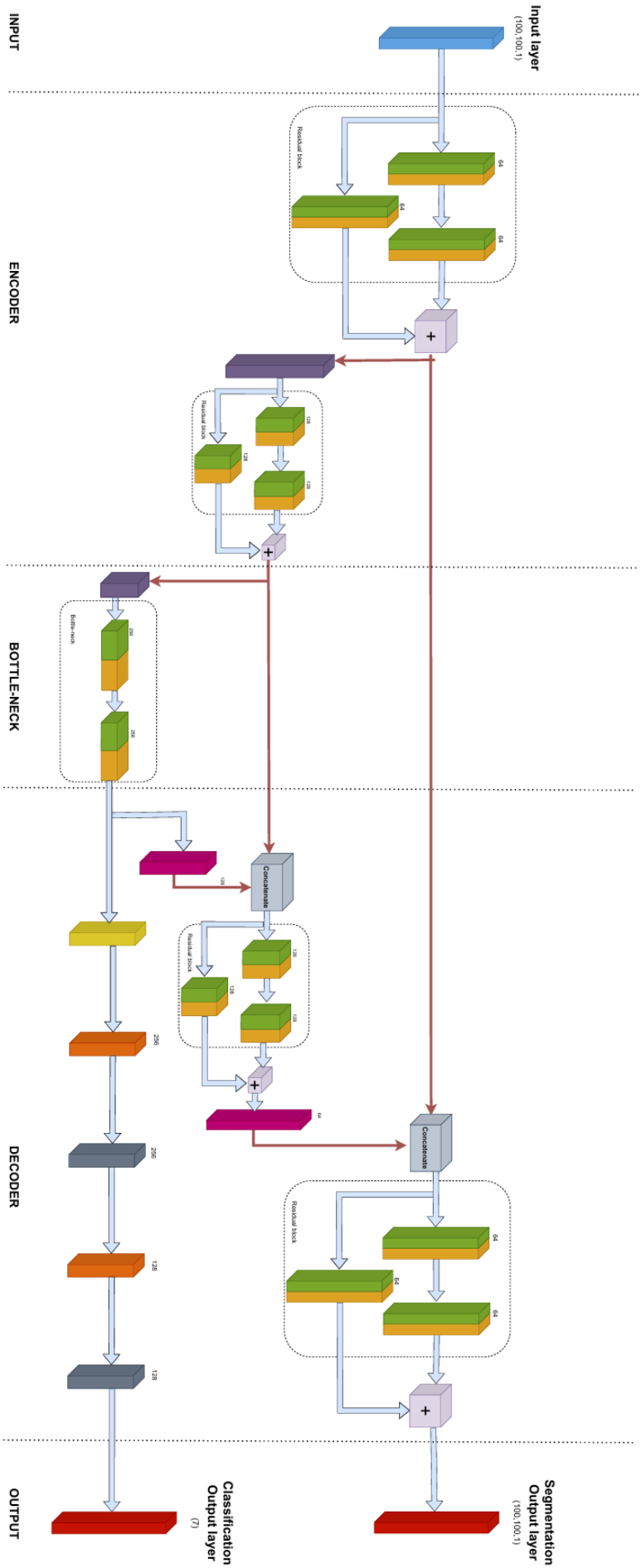


Figure 10: RES-UNET model with 2 Outputs

2.2 Evaluation

... Graphs : (See Fig. 11.).

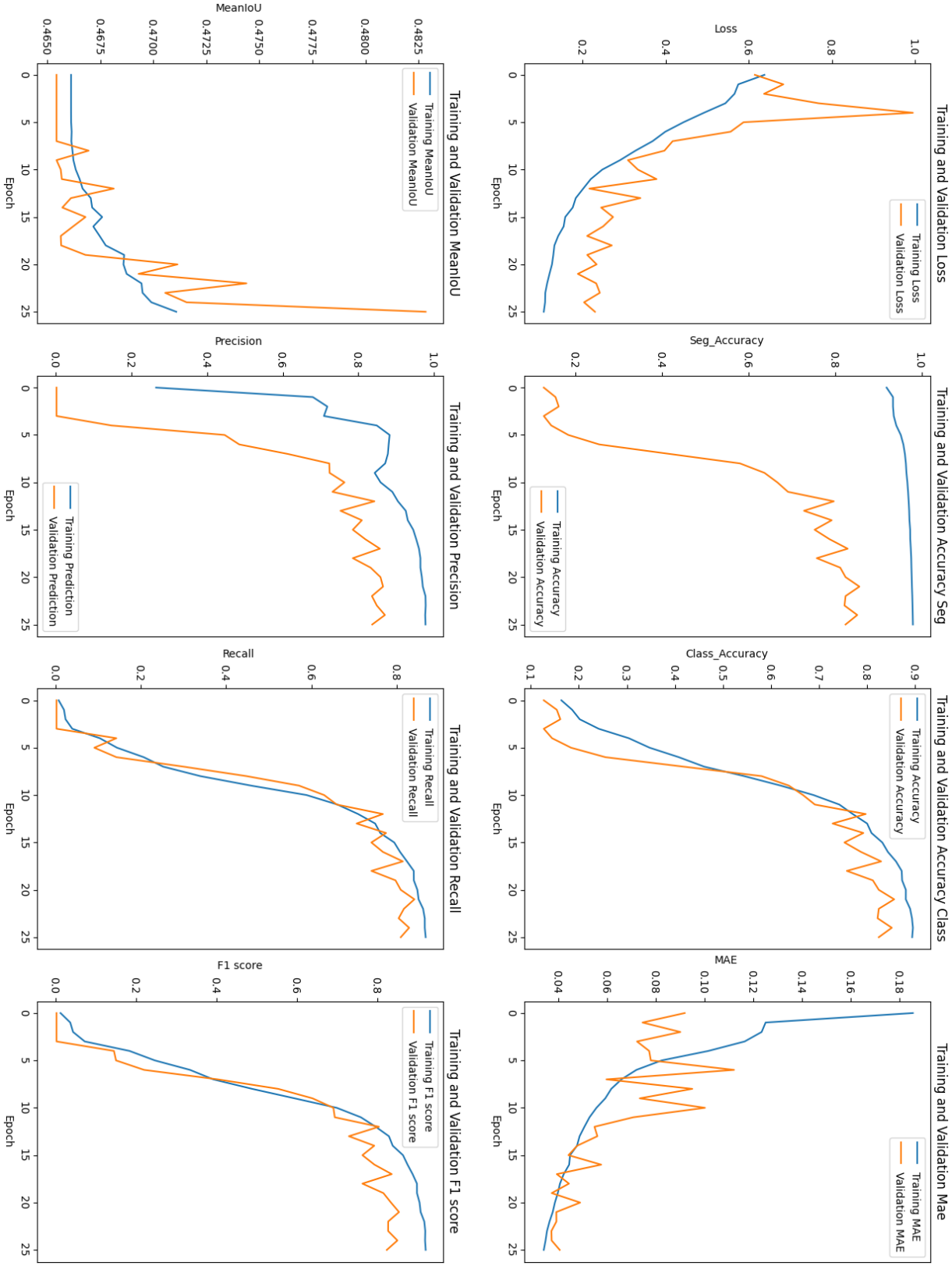


Figure 11: Metrics for Res-Unet model

Confusion matrix : (See Fig. 12.).

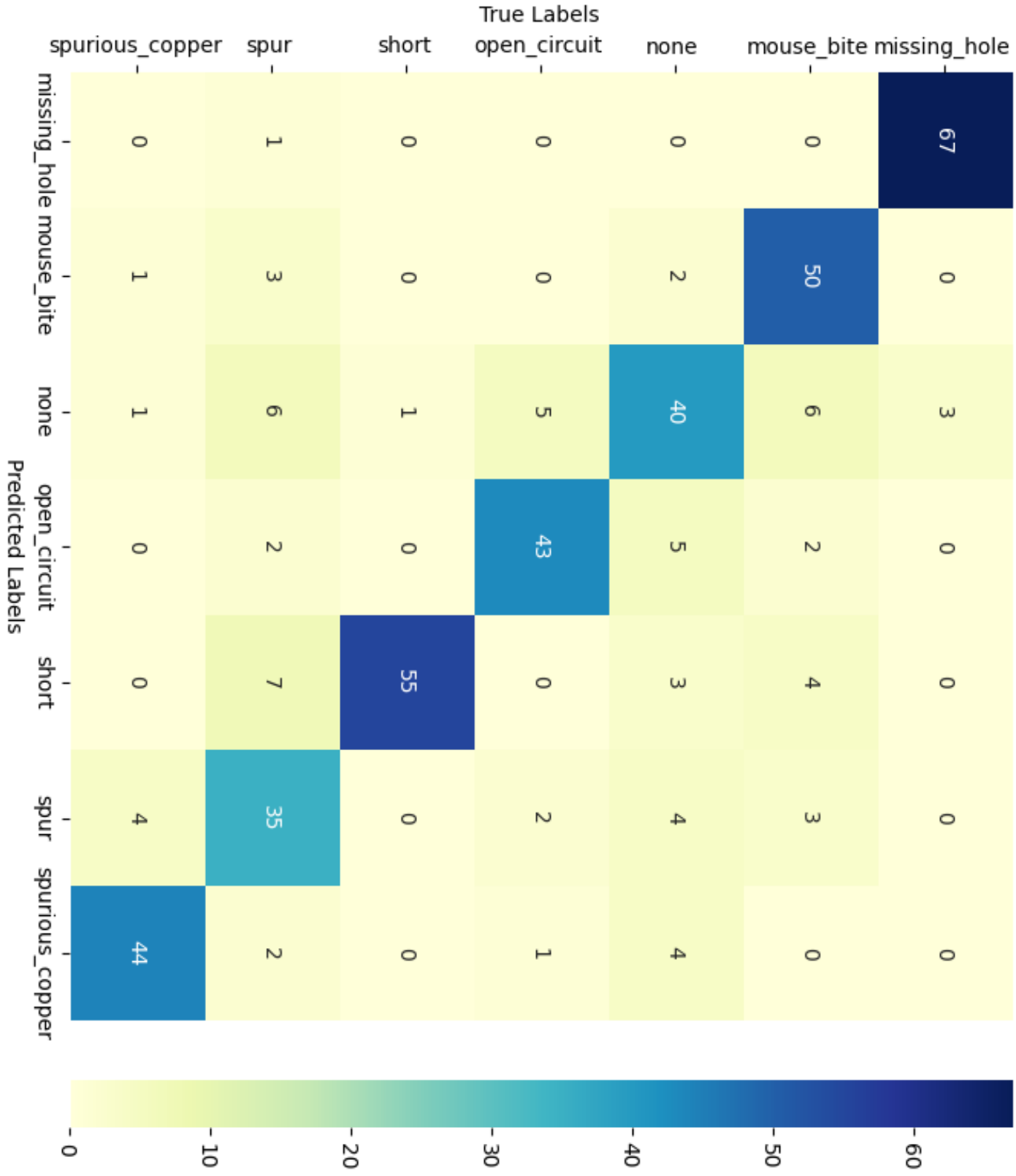


Figure 12: Confusion Matrix for classification output

Results:

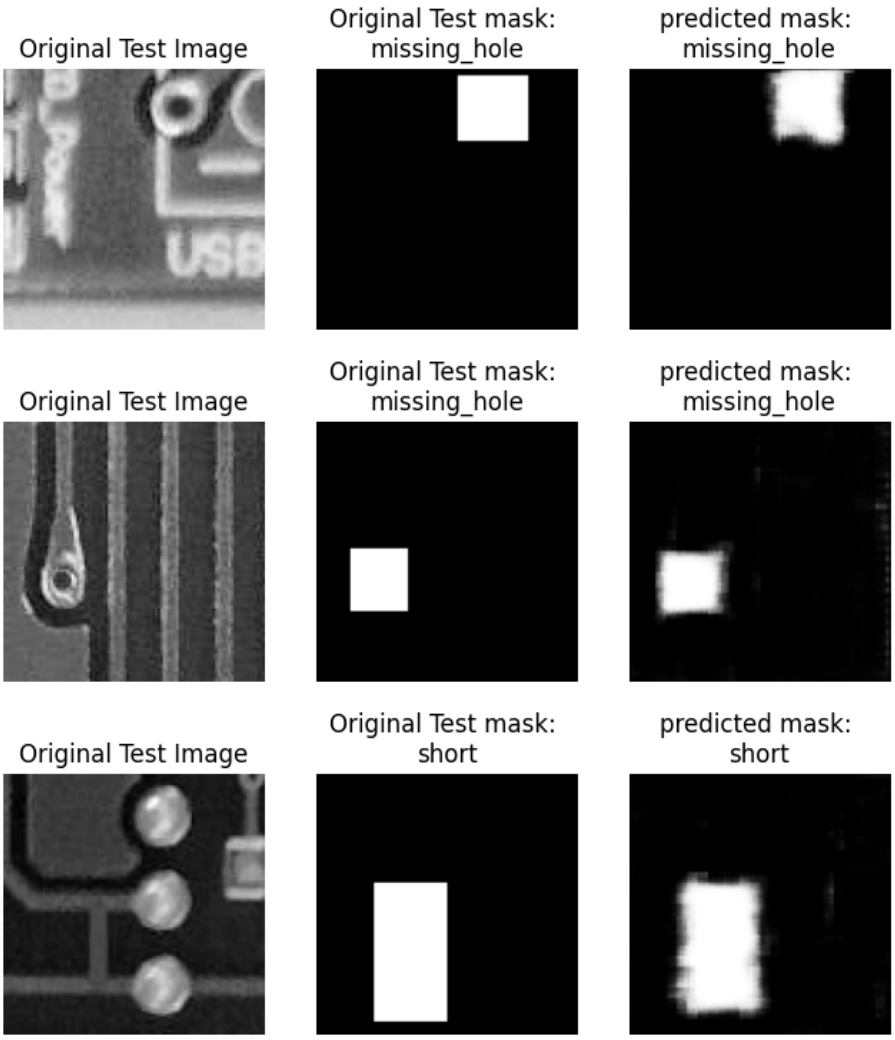


Figure 13: Validation Results

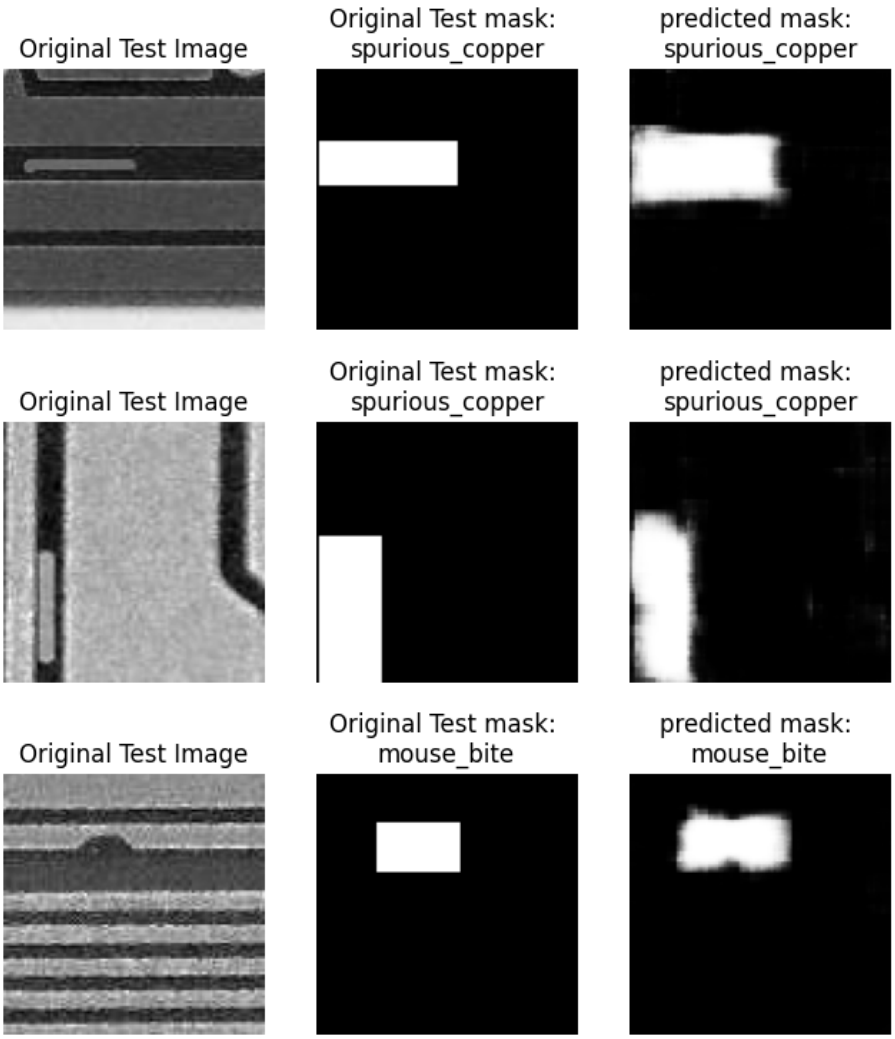


Figure 14: Validation Results

Conclusion

.....

Future Work

We observed many areas of improvement for future scope of work.

- There is potential in doing the machine learning on a computationally powerful computer with GPU. This will help in the context of data-augmentation. At present we had to do a lot of pre-processing to get the small scale images set up for training.
- More metrics can be observed for the current architecture including Dice Coefficient, Precision-Recall Curve, etc.
- Class of defects were limited for this project. Including wafer cracks, copper detachment, component misalignment or damage, etc for future learning will improve model performance in scenarios with real-life defects.
- The image dataset used in this project is very academical in the sense that the PCBs are plain and have no further components attached to them. In a real world application one is interested in defects on the board especially after the attachment of further components to the PCB. This would greatly increase the variety of possible input imagery including shapes and contours our model has not been trained on. Train the model on real world images from PCBs during or at the end of the production process will enable the model to be used in more practical scenarios.

References

[1] Smith, J. (2021). Title of the paper. *Journal Name*, **Volume**(Issue), pages.

[2] Ixiaohuihuihui. (2021). Tiny Defect Detection for PCB. GitHub Repository. Available online: https://github.com/Ixiaohuihuihui/Tiny-Defect-Detection-for-PCB.

[3] Wang, X., Yang, F., Zhang, X. (2022). A novel self-supervised adversarial learning method for defect detection on printed circuit boards. *Journal of Manufacturing Systems*, **63**, 340-352. DOI: https://doi.org/10.1016/j.jmsy.2021.09.006.

[4] Akhatova, A. (2021). PCB Defects. Kaggle Dataset. Available online: https://www.kaggle.com/datasets/akhatova/pcb-defects/data.

[5] Brownlee, J. (2022). How to Perform Object Detection With YOLOv3 in Keras. Machine Learning Mastery. Available online: https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/.

[6] Zhao, H., Lin, Z., Fu, Y., et al. (2020). Residual-Attention UNet++: A Nested Residual-Attention U-Net for Medical Image Segmentation. *IEEE Transactions on Neural Networks and Learning Systems*, **32**(4), 1539-1553. DOI: https://doi.org/10.1109/TNNLS.2020.3011597.