DataScientest

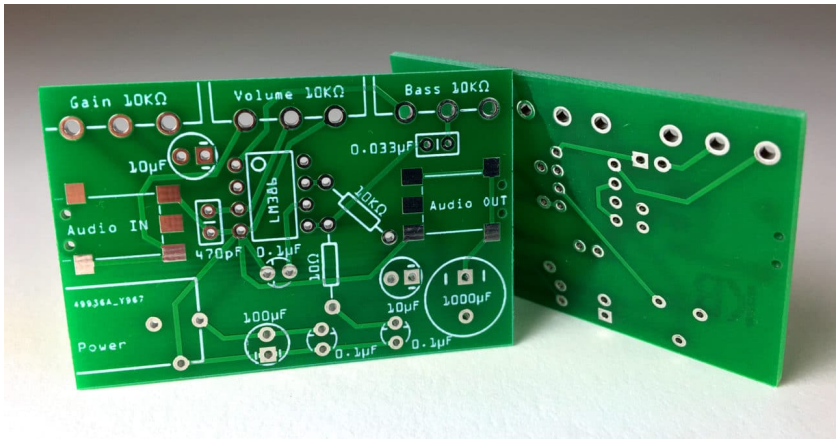
Detection and Classification of Defects on Printed Circuit Boards with Machine Learning

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

Niels Hartano
nylz-ds@proton.me
https://github.com/taubenus

Gernot Gellwitz
g-not@gmx.de
https://github.com/Kathartikon

Supervisor: Gaspard Grimm
gaspard@datascientest.com



https://github.com/wfaiza/PCB_Defects_Detection

June 24, 2024

Contents

Abstract

This report explores various machine learning methodologies for detecting and classifying defects on printed circuit boards (PCBs) using advanced computer vision techniques. Traditional manual inspection methods are time-consuming and error-prone, motivating the adoption of deep learning models such as VGG16[3], RES-UNET[6], and YOLOv5[5] for automated defect detection.

The aim is to enhance the quality control process in PCB manufacturing by leveraging advanced computer vision techniques to observe and identify defects. The study utilizes a large dataset of over 10,000 annotated PCB images, encompassing various defects. Rigorous preprocessing, including data augmentation and model training, demonstrates promising results in each architecture's performance.

The manually designed RES-UNET model achieves high accuracy in defect segmentation and classification tasks, while YOLOv5 also demonstrates efficient defect detection capabilities. Future work involves enhancing model performance through extended training epochs and refining interpretability techniques.

Overall, this project contributes to enhancing quality control processes in PCB manufacturing and refurbishing, ensuring higher precision and efficiency in defect detection.

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 time consuming tasks for PCB manufacturing companies [1]. 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 [1].

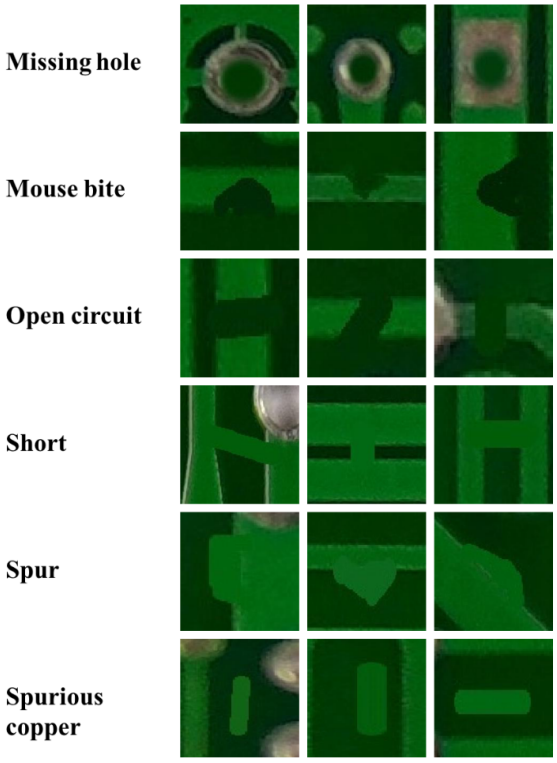


Figure 1: Sample defects explored in this project

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 three 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 RES-UNET model (UNET including a residual connection block hence RES-UNET model). While the third is based on YOLOv5 implementation. VGG16 makes use of so-called bounding boxes to detect defects, while the RES-UNET model uses mask segmentation (images) for detection and labels (label-encoded) for classification of defects. YOLOv5 also takes as input the encoded labels and bounding boxes coordinates for detection (segmentation and classification).

VGG16 can be downloaded as a pre-trained model from the TensorFlow Keras library, enabling transfer learning. In contrast, we developed and implemented the RES-UNET model from scratch. Similarly, YOLOv5 is available for download from Ultralytics. Leveraging these pre-trained models (VGG16 and YOLOv5), which have been trained on extensive datasets, allows us to harness their pre-trained weights for training on our dataset, resulting in tangible improvements and robust performance.

A public PCB dataset containing over 10,000 images with 6 kinds of defects (See List. 1.) 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.

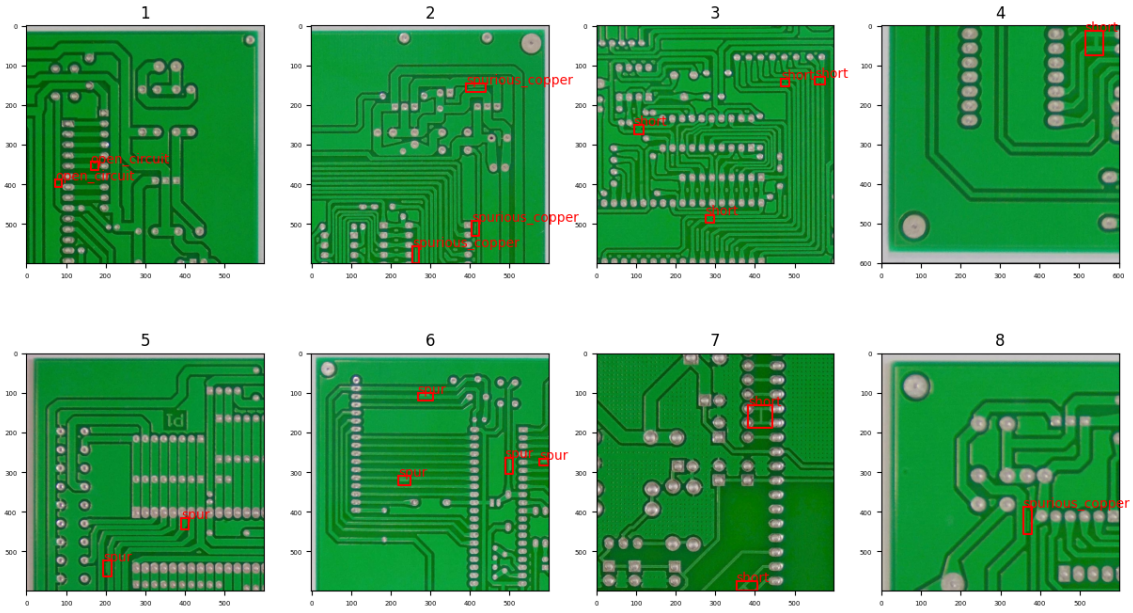


Figure 2: Sample images from the PCB dataset with annotated defects

Some images only contained one defect, while others contained multiple defects (See Fig. 2.), but the class of multiple 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 (some images had a single defect while others had up-to 5 defects). The defects appeared to be fairly distributed in the dataset, with the most common defect being ”Mouse Bites” and the least common defect being ”Shorts” (See Fig. 4.).

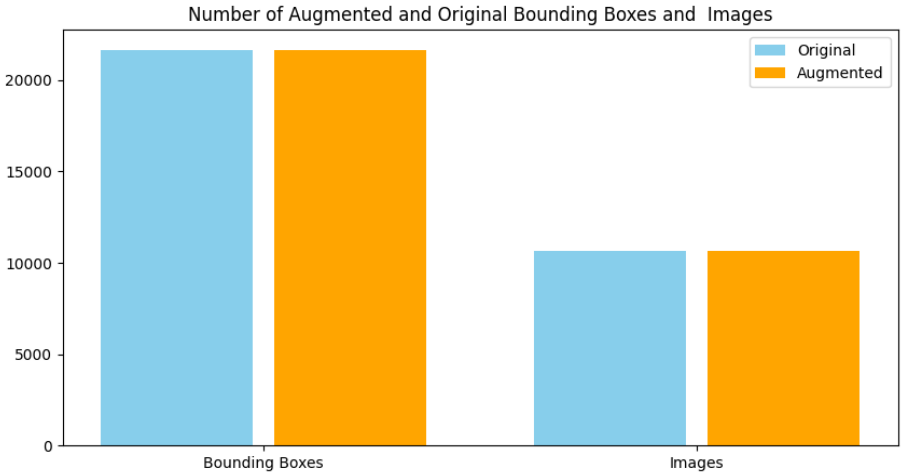


Figure 3: Ratio of Number of defects to Number of images for VGG16 model

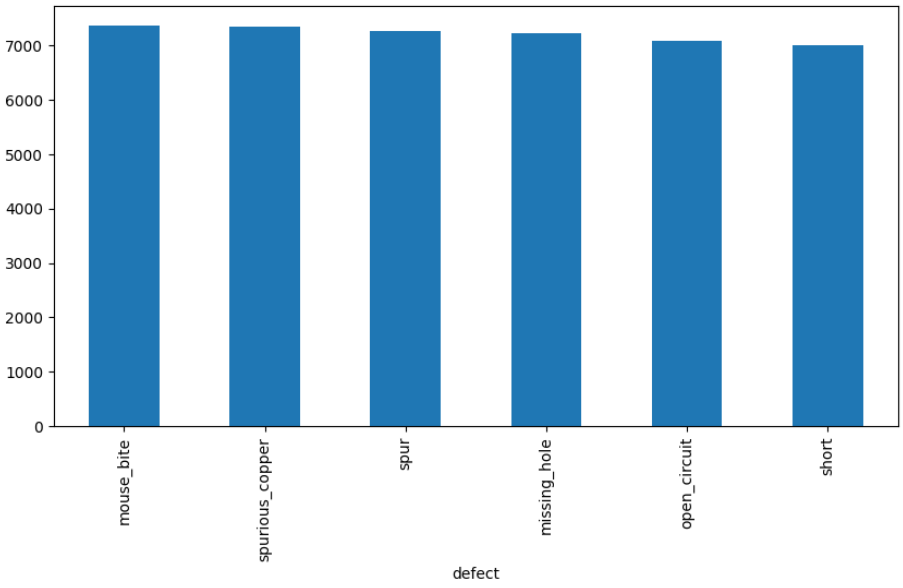


Figure 4: Defect distribution

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 potential 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 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[7]. 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[7]. Two approaches were considered for data augmentation: Using a library or implementing the augmentations manually. 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"[8], 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

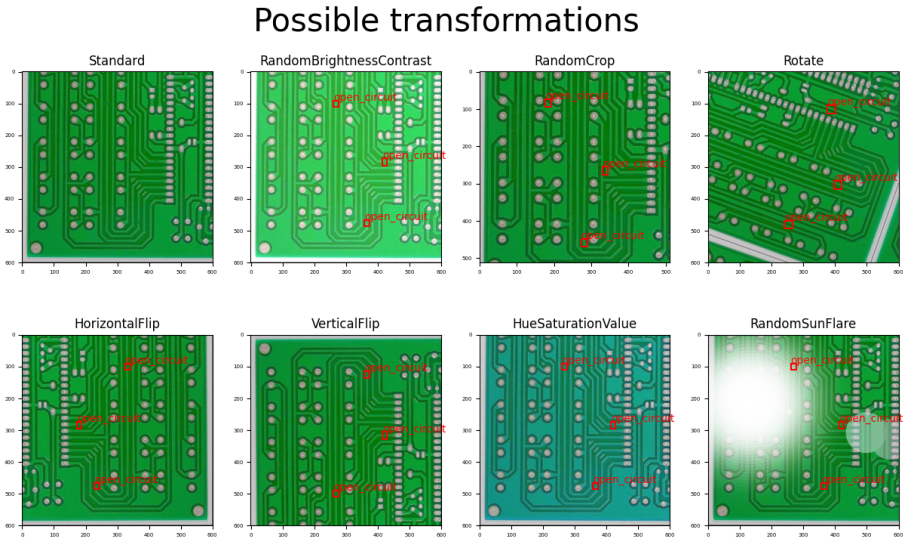


Figure 5: Possible augmentations by ”Albumentations”.

cropped part of the image, which is not desired. In this case the defect would not be detected 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.

A major 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 image s[9] (See Fig. 6.).

Another decision made early on after observing the size of images (dim:600x600), was to crop the images to easily processable dimensions i.e. (dim:100x100) [9] (See Fig. 7.).

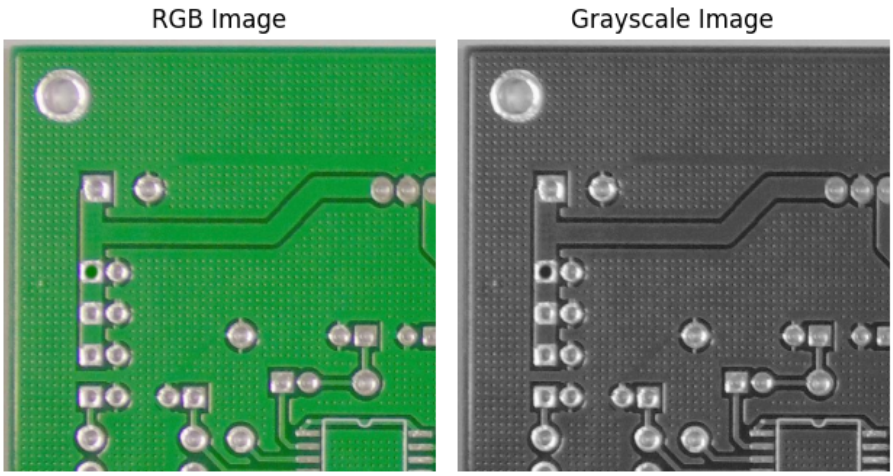


Figure 6: Colored vs. Grayscale image

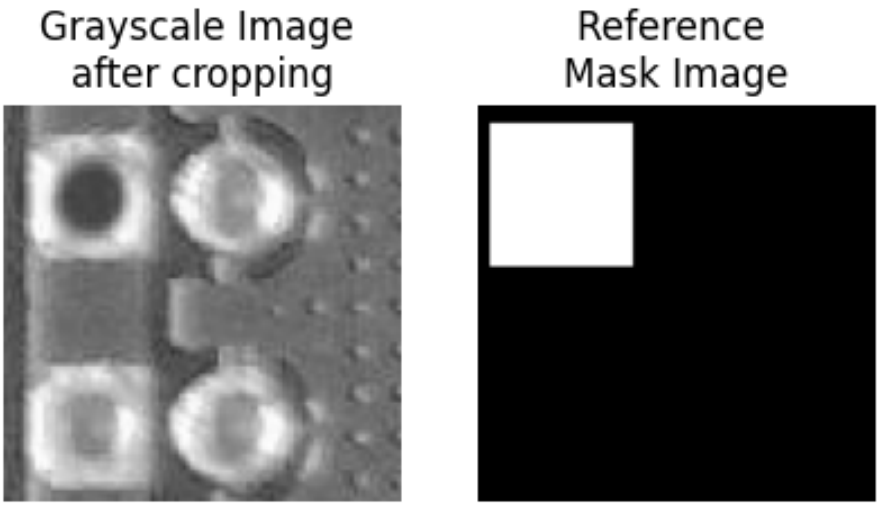


Figure 7: Cropping image and mask to 100x100 dimension

We observed various issues during the process of implementing on-the-fly augmentation by data generators like ImageDataGenerator[10] on the dataset. We managed to solve some of those issues; 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. Similar to Albumentations; the augmentation provided by ImageDataGenerator performed rotation and zoom transformations differently on 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 model, the following augmentation techniques were implemented (See Fig. 8.).:

- Rotating 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 to the image and its corresponding mask. In our case, we had to introduce the ”none” class to offset the generation of cropped images that had no ”defects”.

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 losses. 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.

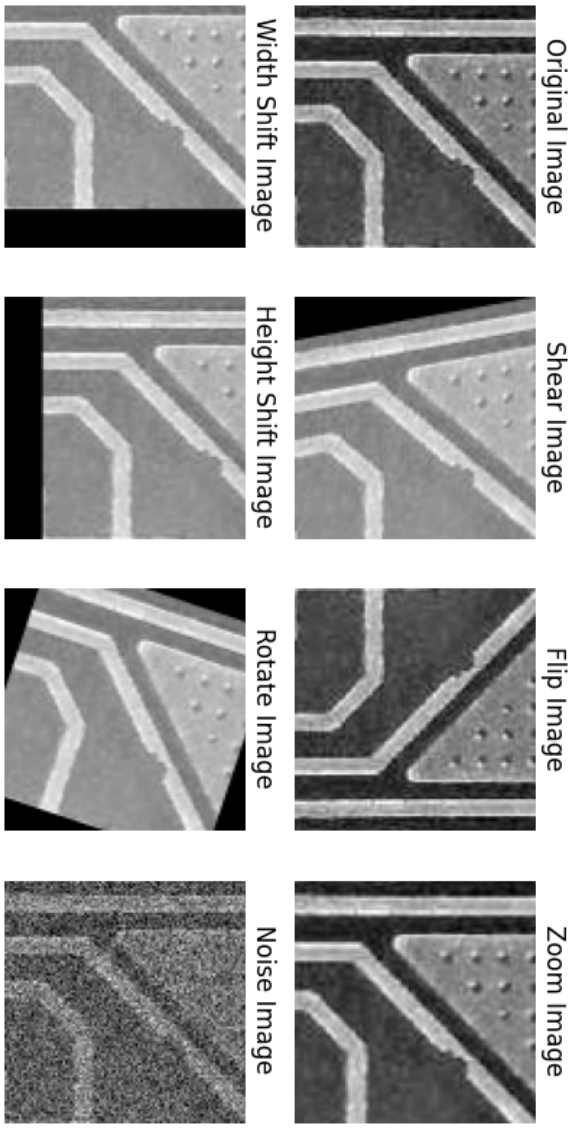


Figure 8: Manually implemented augmentations

Modelling

The methodology of this project involves several key steps: data set generation and preprocessing, model design and training, and evaluation. As discussed above; the dataset generation and preprocessing was handled with the help of "Albumentations" and manually. Now we will discuss the architecture design and modelling of the various image processing models.

2.1 Model Design

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 (See Fig. 9.). The model has been pre-trained on the ImageNet[4] 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[11] 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.

By integrating separate pathways/modules, different loss functions and metrics can be applied to each task. Hence, our implementation enabled us 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 [19].

Unfortunately, the input for VGG16 can only handle single bounding boxes per instance. And in the dataset there are multiple instances with 2 or more defects. Hence, due to the pre-processing error, the model could not produce the desired results at the moment. We are currently still trying to get this implementation to work. So hopefully in the future results will be shared once this model is successfully implemented.

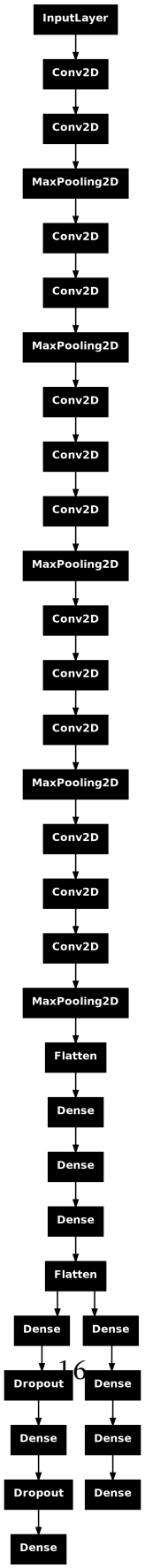


Figure 9: VGG16 with two pathways/modules.

2.1.2 RES-UNET

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 U-net 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.

Using two different branches/modules enables us to employ different loss functions (while emphasizing the loss weights for the training) and metrics for the detection and classification tasks.

We also utilized the various callbacks (timing, early-stopping, reduce-learning-rate and checkpoint) to avoid over-fitting and over-computation by unnecessary training.

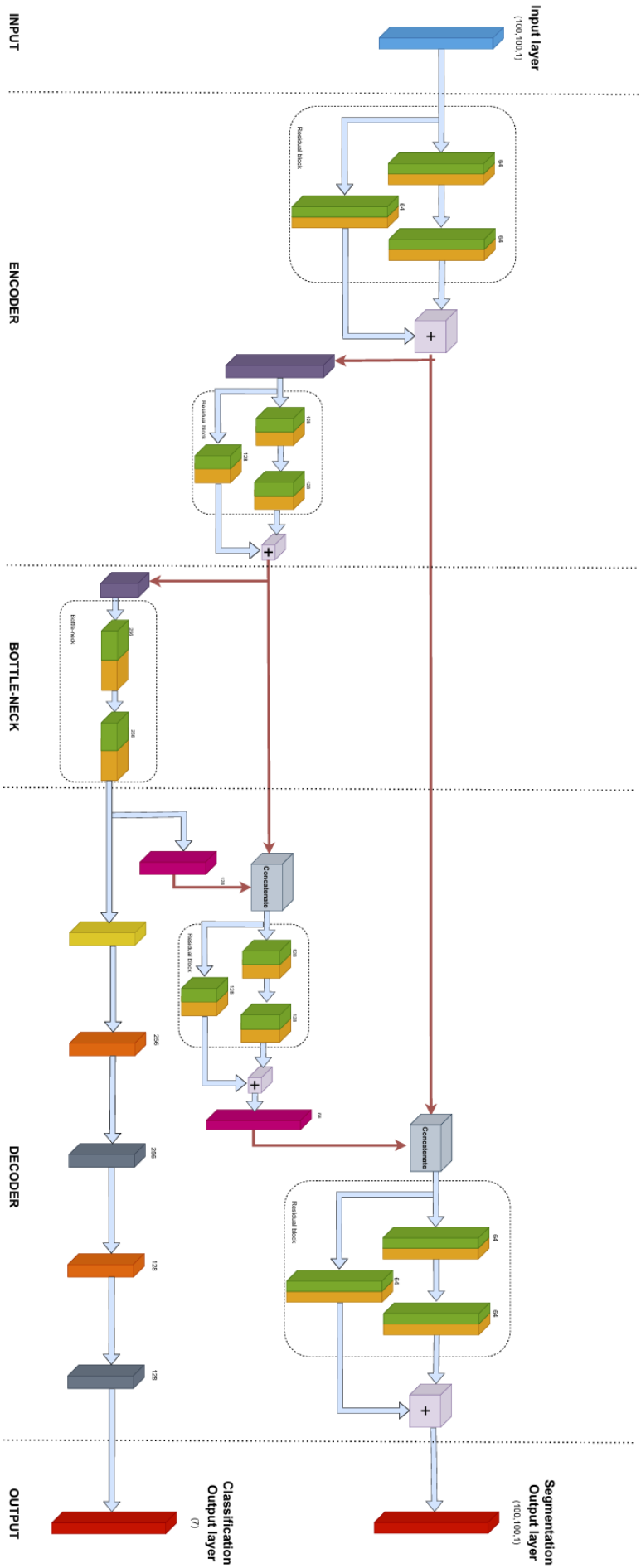


Figure 10: RES-UNET model with 2 Modules/Outputs

2.1.3 YOLOv5

In addition to designing and developing our model for training, we also successfully implemented the YOLOv5 object detection model developed by Ultralytics on the PCB dataset. Unlike the original YOLO models built on DarkNet, YOLOv5 is developed with PyTorch, which enhances ease of understanding and usage.

This model can be utilized for both segmentation and classification, providing us with the opportunity to compare the results of our model with this pretrained and well-established design.

YOLOv5 uses input images in RGB format with a resolution of 640 pixels, which suits our dataset. Our image data has a resolution of 600 pixels. YOLOv5 provides the feature of passing images directly to the model for image augmentation automatically. During testing, it was observed that passing the images as-is (600 pixels) did not provide optimal results. Therefore, the images and corresponding labels were padded to resize them to 640 pixels to get the best results.

For reference, the YOLOv5 model architecture is elaborated in Fig. 11. For more details on model architecture please go to:

https://docs.ultralytics.com/yolov5/tutorials/architecture_description

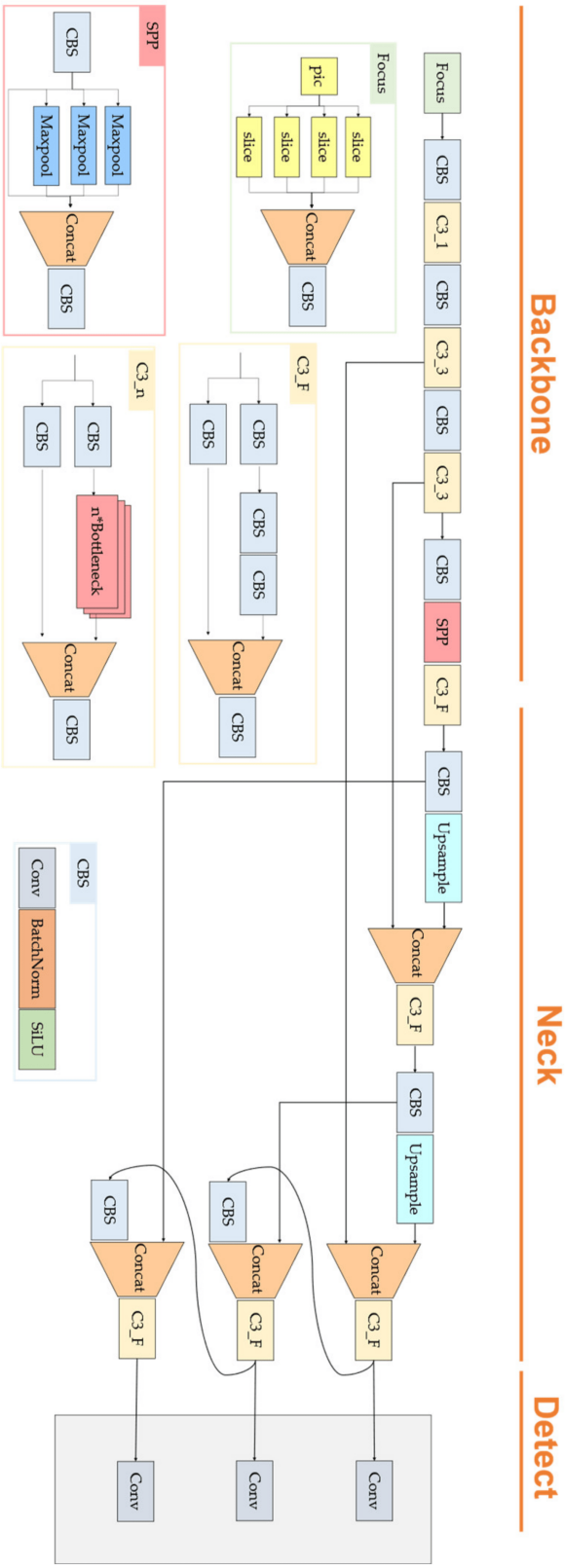


Figure 11: YOLOv5 model architecture from [17] available under CC BY 4.0.

After the required data augmentation, the model can start training with the following command:

```
!python path/to/train.py --data /path/to/data.yaml --weights
↪  yolov5s.pt --img 640 --epochs 50 --batch-size 16 --cache
```

In the current iteration of our training, we used the default Confidence and IoU thresholds (conf-thres=0.25 and iou-thres=0.45). The model was trained for 50 epoch, with batch size as 16. The pretrained small model (yolov5s.pt) weights were used. Potentially with further fine tuning training it can be done by freezing some layers and training on the desired layers. The choice to train on small model is because of the computational restrictions of our personal computers and also time restrictions[5]. The results were compelling enough that it presents a good solution for our current object detection project. The output results for the models are discussed in the next section.

Evaluation

The performance of the models was evaluated using accuracy, MeanIoU, precision, recall, and F1-score metrics. The VGG16 model unfortunately could not achieve any results but for the YOLOv5 and the RES-UNET model we achieved a classification accuracy of 95%.

Despite the high accuracy, both models faced challenges in detecting certain defects. The RES-UNET model struggled with detecting small defects, while the YOLOv5 model had difficulty distinguishing between similar defect types. The results are discussed in detail below.

3.1 RES-UNET

The **size of our validation** set is 20% of the cropped, separated, balanced and pre-augmented original training set. This means there are 413 validation samples. The evaluation metrics are based on these samples. The defect ratios can be seen in Fig. 12.

Defect	Count
Missing Hole	65
Mouse Bite	55
Open Circuit	59
Short	57
Spur	65
Spurious Copper	47
None	65

Figure 12: Validation Set Composition

The graphs in Fig. 13 show the development of **key metrics during the training** epochs. We used Binary-Cross-Entropy and Categorical-Cross-Entropy as loss functions for the segmentation and classification outputs, respectively. The respective activation functions have been Sigmoid (binary) and Softmax (multi-class). Validation accuracy for segmentation is at 98.0% after the training with 14 epochs with loss weights 80/20 seg/class.

We can observe some encouraging results from the validation of the RES-UNET model. The metrics' graphs show improvement still which indicates that

the model will improve with training over more epochs.

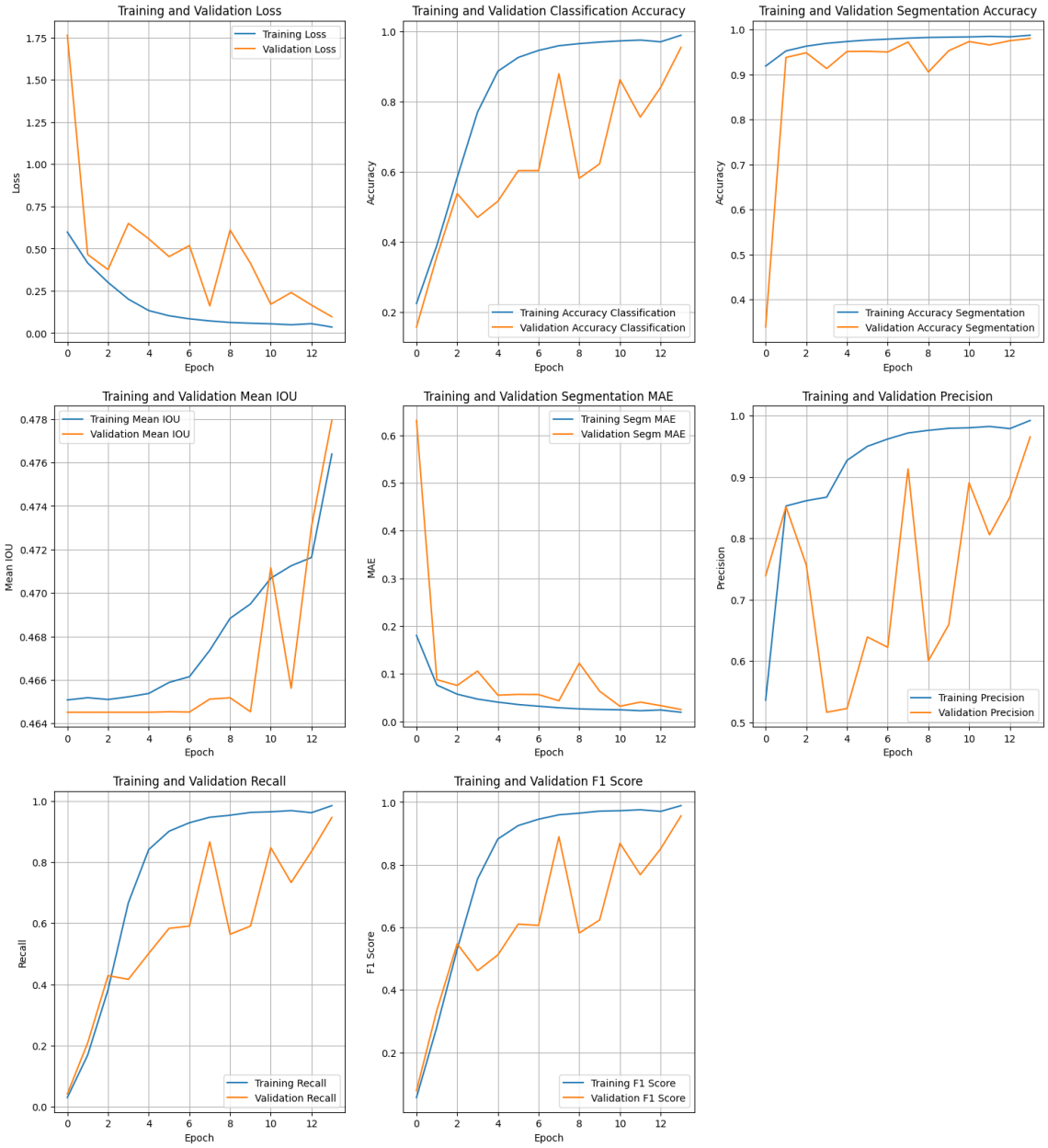


Figure 13: Metrics for RES-UNET model

The **classification report** (See Fig. 14.) and the **confusion matrix** (See Fig. 15) on the validation set show that precision and recall for each defect class vary but they are generally stable. 100% of the short circuit detections on the test set are correct, while still 15% of the detected ”none” class are from a different, indeed all not-none classes are erronuously predicted as ’none’ at least once. Likewise, 98% of the missing holes are correctly identified, while 8% of the ”none” and ”spur” class are not being detected. Hence we can observe that the defect classes of ”none”, ”spur” and ” mouse bites” are the most difficult to detect accurately. Especially the ’none’ class should have a high precision whereas a suboptimal recall on that class is more tolerable. We are interested in finding the defects on the PCBs, so one would need to have as few wrongly detected ’none’ classes as possible. Meanwhile a wrongly classified real ’none’ class is just ’false alarm’.

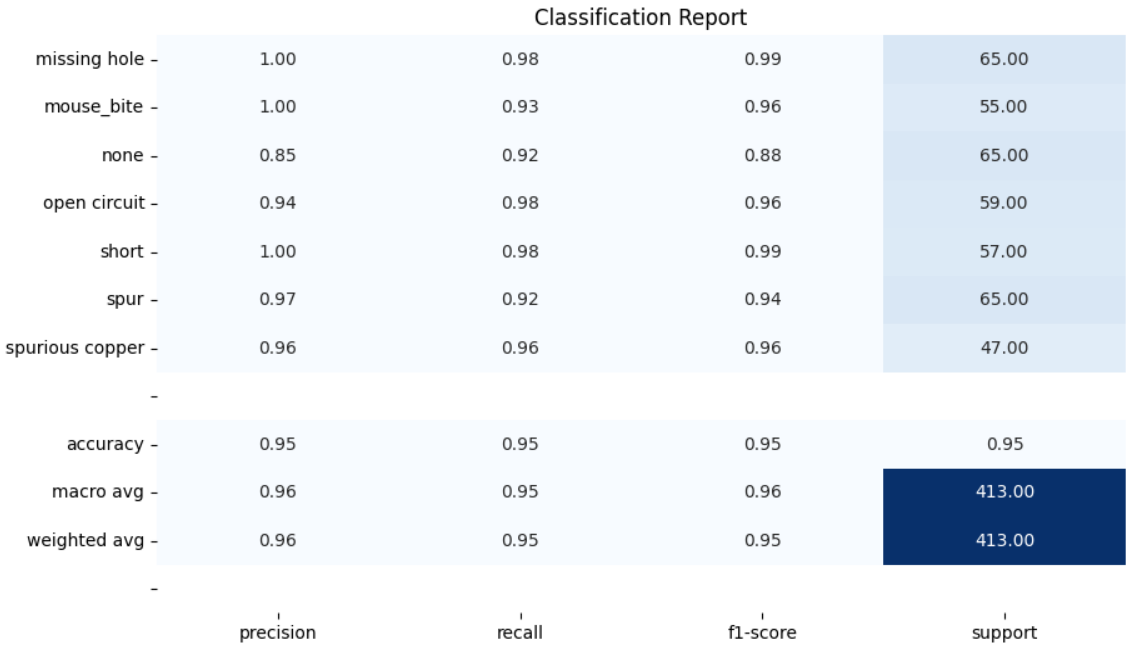


Figure 14: Classification metrics for classification output

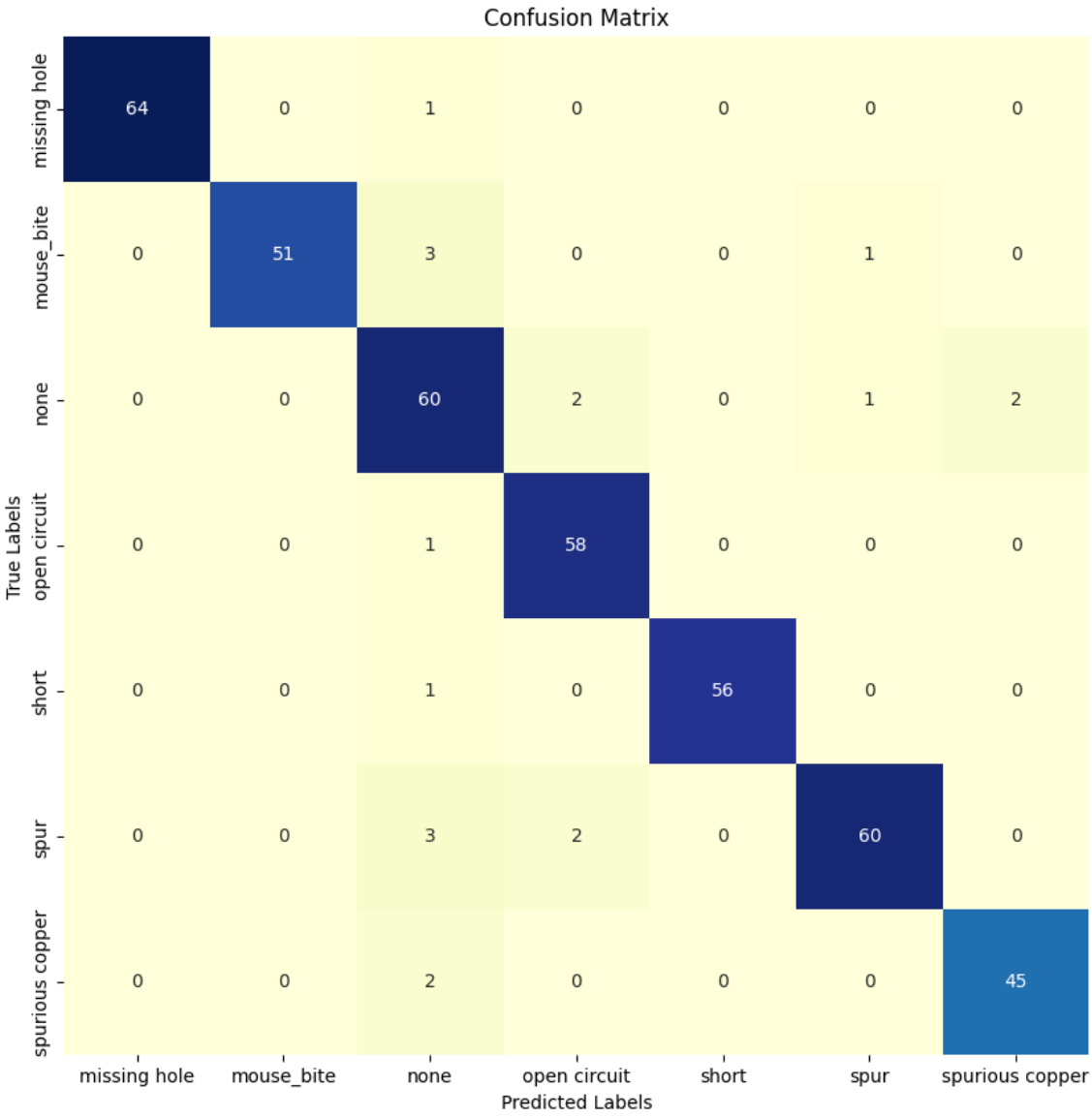


Figure 15: Confusion Matrix for classification output

3.1.1 RES-NET model Results

Fig. 16 and Fig. 17 illustrate that the location of the defects or the pixel matrix is predicted quite precisely. For clarification the real and predicted classes are shown.

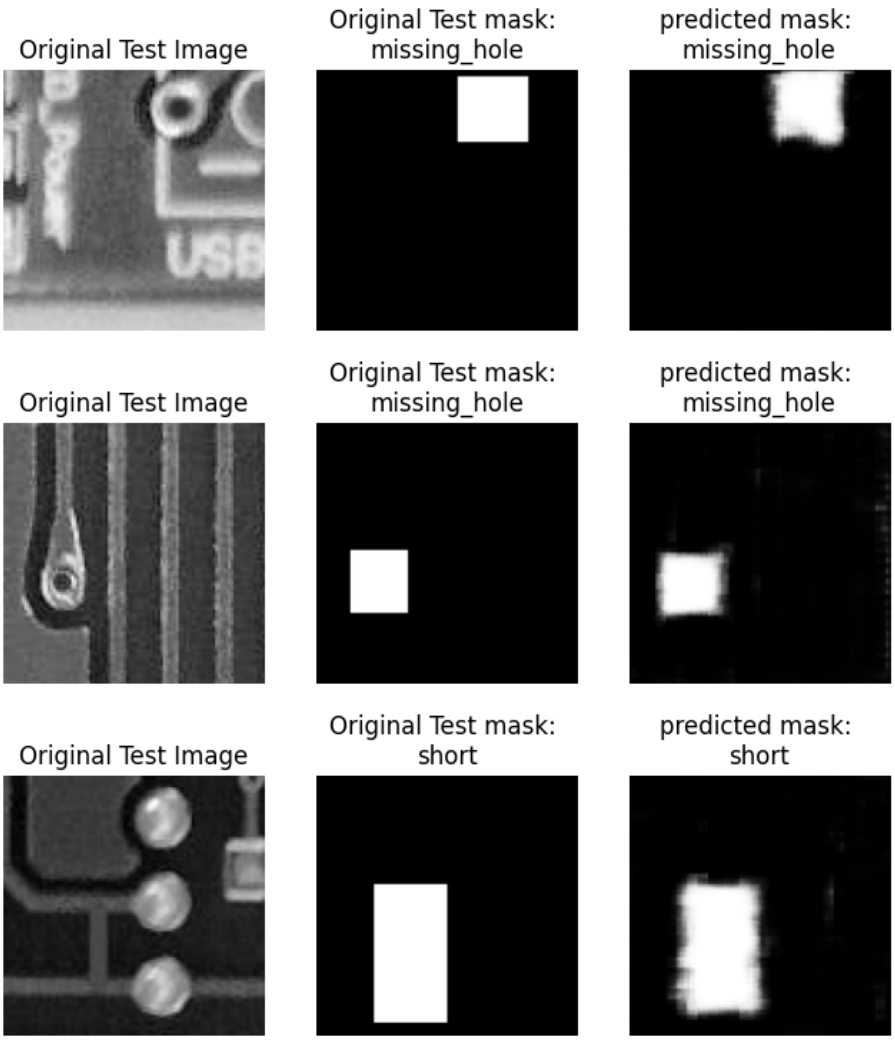


Figure 16: Validation Results

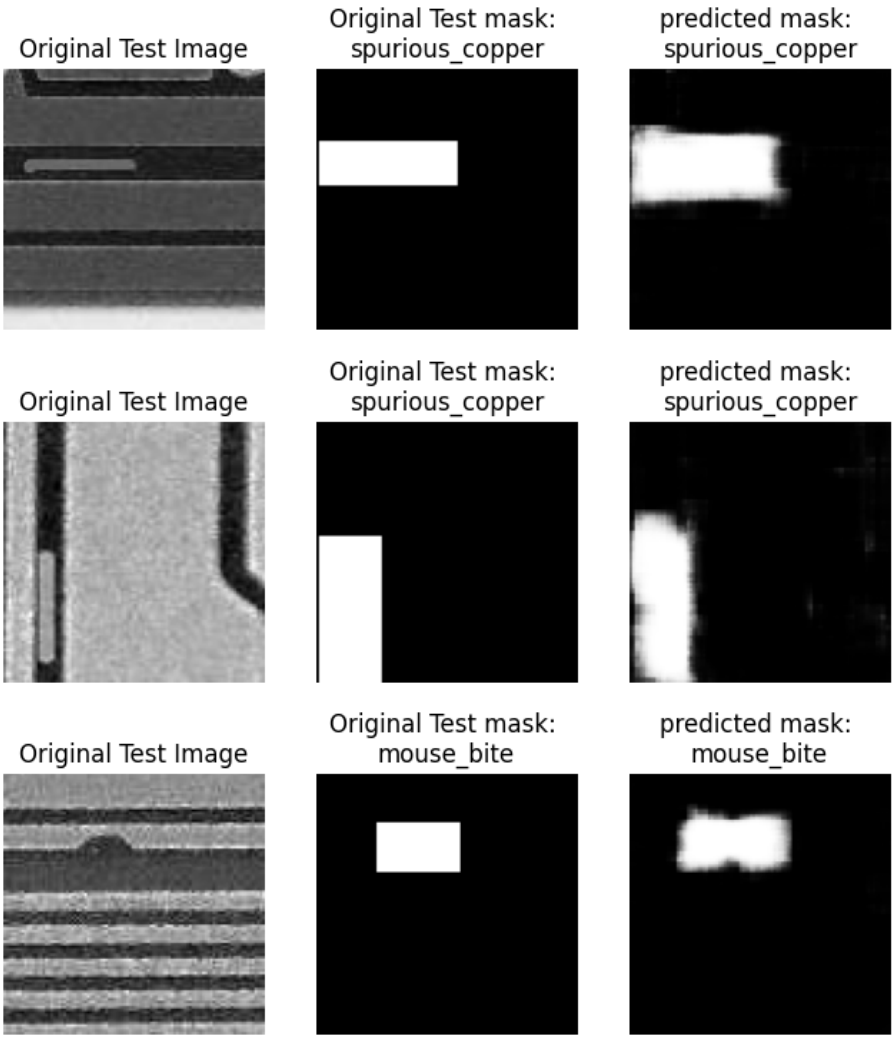


Figure 17: Validation Results

Fig. 18 presents the final representation of the results obtained from the image processing. The displayed results demonstrate a high degree of accuracy and precision, highlighting the effectiveness and potential of this machine learning model.

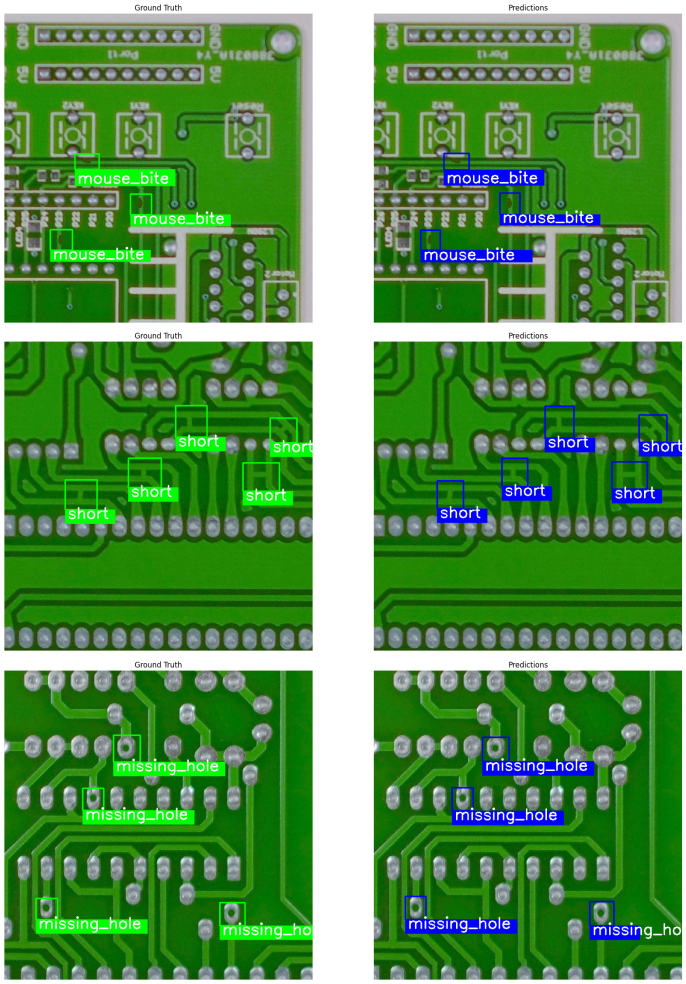


Figure 18: Final prediction Results for RES-UNET model

3.2 YOLOv5

The model was trained for 50 epochs and it can be observed from the resulting graphs that the training is still improving hence for further tests we can train the model over more epochs, though by observing the fluctuations in the validation loss, we can argue that model might be moving towards over fitting. The results are observably accurate already with just ”Mouse-bite” and ”Spur” type of defect giving issues to the model. The ”Missing-hole” defect is the easiest for the model to detect, compounding the observation from the U-net model results which also performed the best for this type of defect. The model was trained on pretrained weights for YOLOv5s since this is a small to medium size dataset. In future, training with fine tuning self-initialized weights can improve the results especially in case of over fitting.

Graphs for YOLOv5 model : (See Fig. 19.).

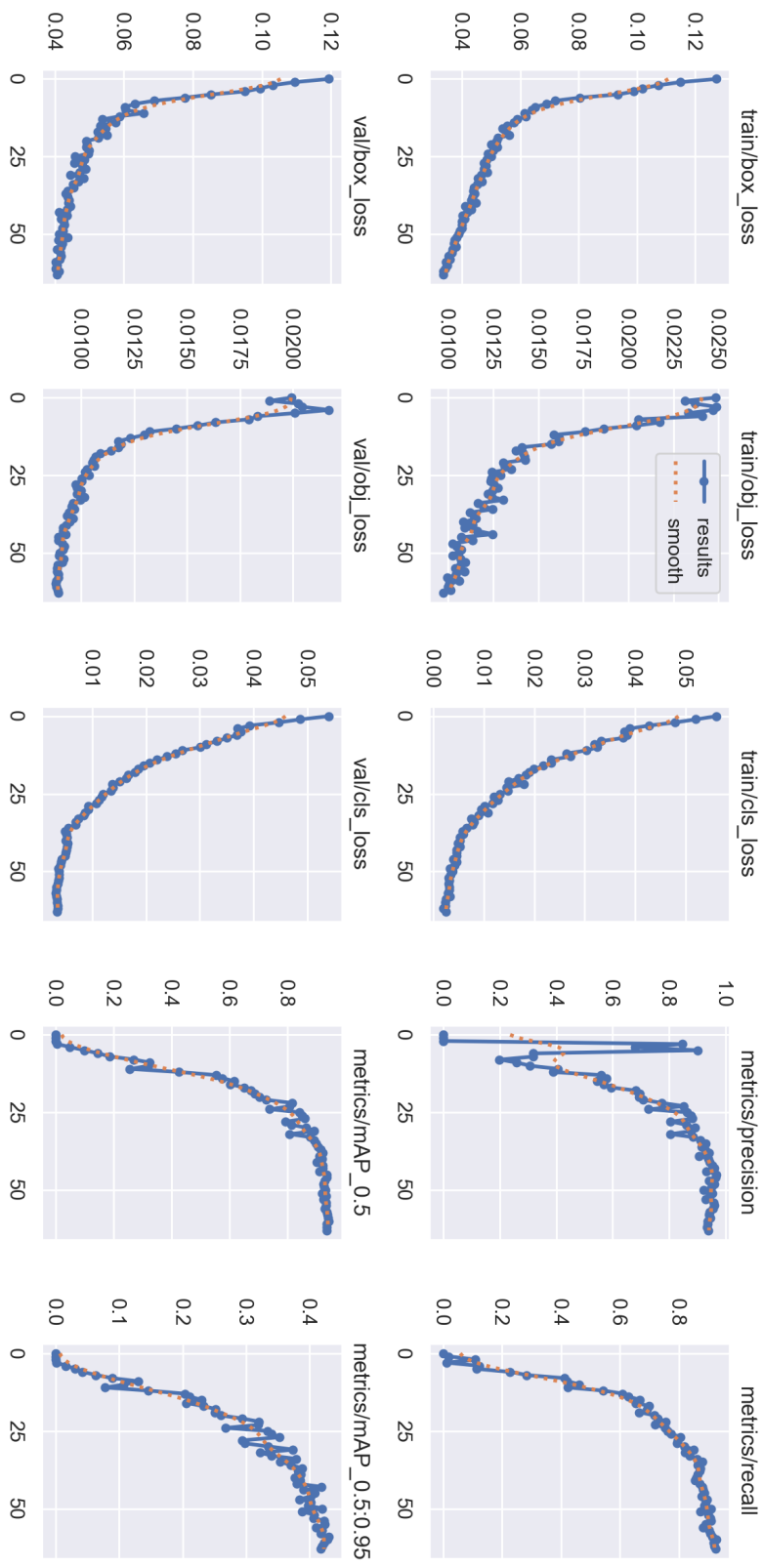


Figure 19: Metrics for RES-UNET model

Confusion matrix : (See Fig. 20.).

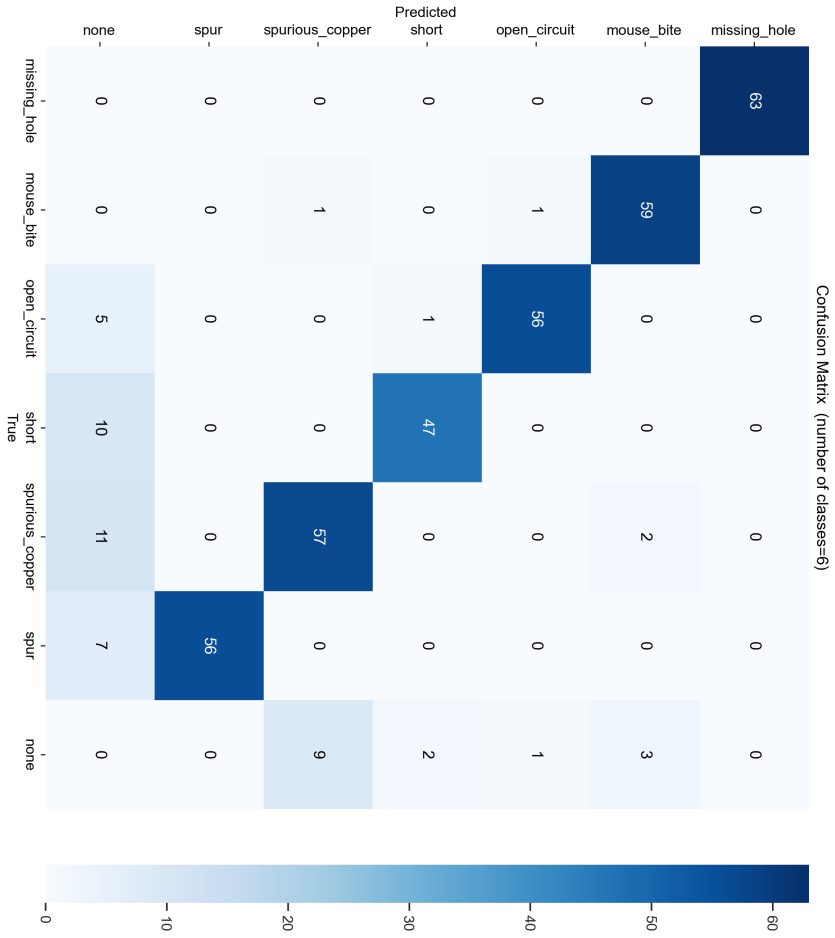


Figure 20: Confusion Matrix for classification output

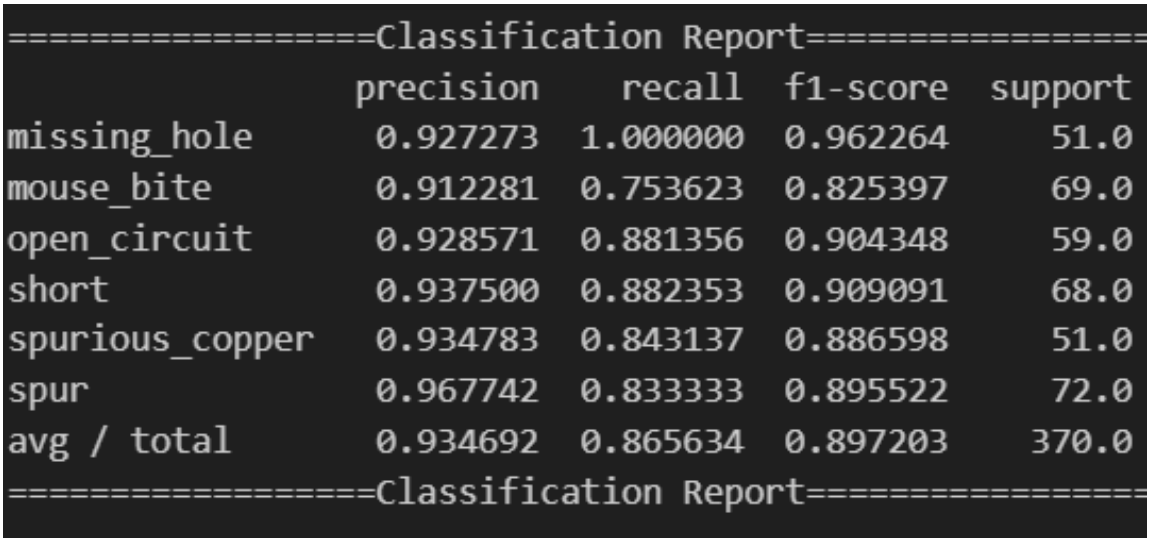


Figure 21: Classification metrics for classification output

3.2.1 YOLOv5 model Results

The results for prediction on validation data can be seen in the following figures. The results are very accurate, but there are still undetected defects e.g. in Fig. 22, the open circuit image has 2 defects whereas only one was detected.

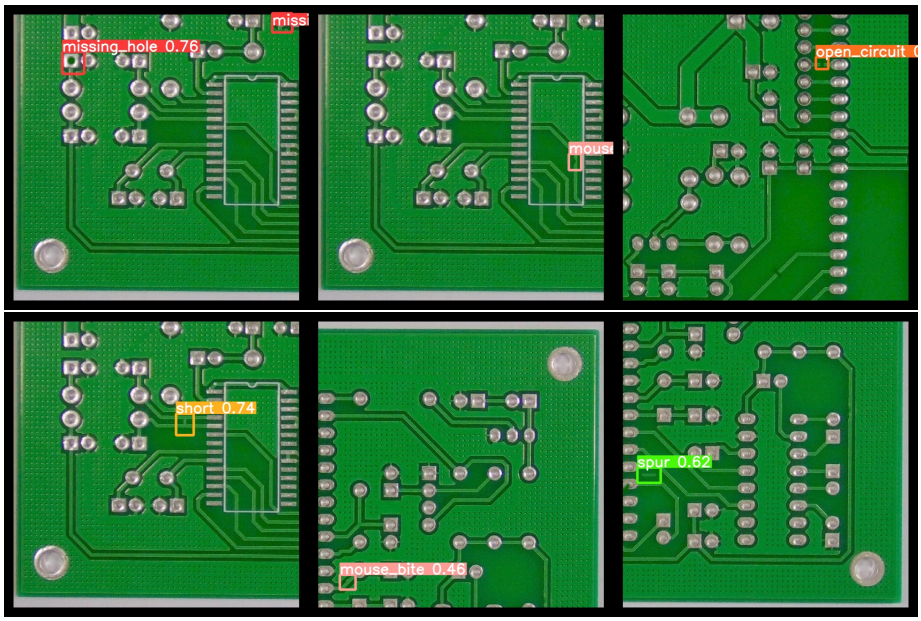


Figure 22: Validation Results

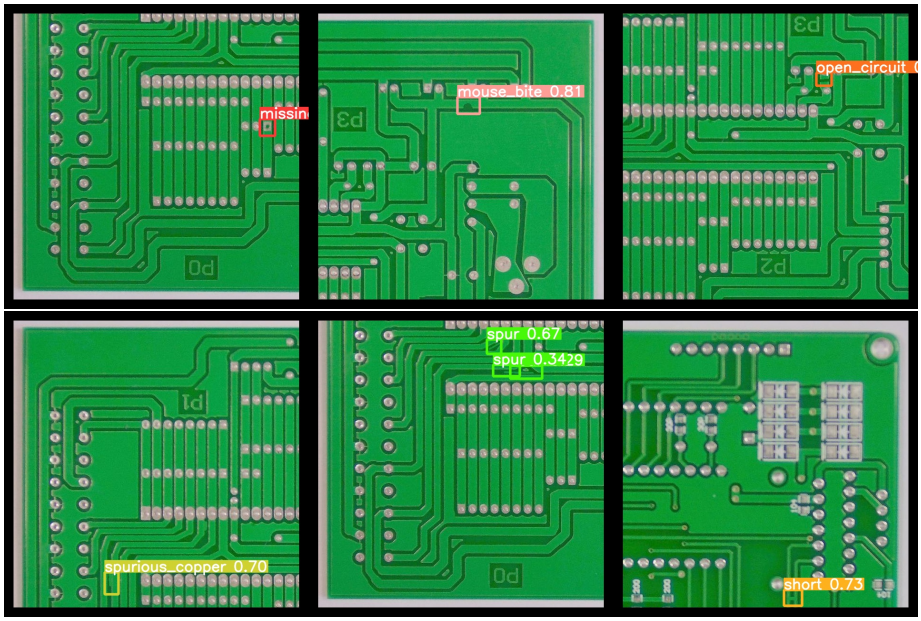
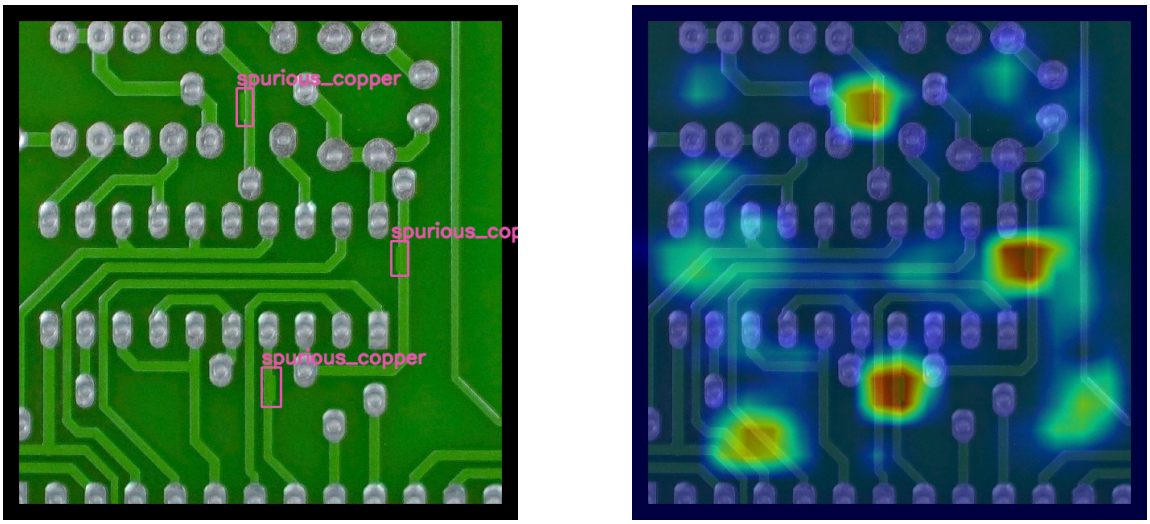


Figure 23: Validation Results

3.3 Model Interpretability

Model interpretability is crucial for understanding the decisions made by deep learning models, especially in complex tasks like image segmentation and classification. Since our project was a comprehensive image detection project(segmentation and classification); we had to implement an approach for understanding the behavior and decision-making process of the deep learning model. Implementing such behavioural interpretation can help understand the strengths and weaknesses of the model and guide us in improving its trustworthiness and reliability.

3.3.1 Grad-Cam



(a) Original image (b) Grad-Cam Segmentation Result

Figure 24: Grad-Cam results for YOLOv5

We implemented Grad-Cam for the YOLOv5 model. Keeping in mind that this is not a comprehensive study on the model's interpretability and therefore does not reflect the results obtained for prediction accurately, the results obtained provide a good outlook for the model behaviour. Model interpretability is a massive field of study that can be further explored in the future for a comprehensive understanding of the model behavior.

Conclusion

The machine learning approach for defect detection and classification on PCBs demonstrates significant potential for improving the quality control process in PCB manufacturing.

4.1 RES-UNET architecture

While the classic U-NET architecture was sufficient to achieve already high metric values in the segmentation, it was not before we combined it with residual blocks in each encoding and decoding step of the UNET that the accuracy of the classification branch moved to acceptable levels. In fact, the RES-UNET model demonstrated high accuracy in detecting and classifying PCB defects, proving to be a robust solution for PCB defect detection. The combined architecture of Residual Networks and UNET provided the necessary features for precise defect segmentation and classification, with a segmentation and classification accuracy of about 97% and 95%, respectively, on our chosen dataset. This evaluation highlights the successful application of the RES-UNET model, providing a strong foundation for further enhancements and deployment in real-world PCB inspection scenarios.

Hence we can conclude that the model works very well on our chosen dataset.

4.2 YOLOv5 architecture

The YOLOv5 model, with its high accuracy and efficiency, shows promise for efficient defect detection. Future work may focus on enhancing the models' performance by incorporating fine tuning training weights, improving data augmentation techniques, and exploring other advanced interpretability techniques.

Future Work

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

- The training metrics indicate that additional epochs could yield better results. Due to computational and time constraints, we limited each model to 50 epochs. A key improvement would be to train or retrain the models for 100 epochs.
- There is potential in leveraging a computationally powerful computer with a GPU for machine learning which can significantly enhance data processing. Currently, extensive pre-processing is required to prepare the small-scale images for training.
- More metrics can be observed for the current architecture including Dice Coefficient, Precision-Recall Curve, etc.
- For training the RES-UNET model, the augmented training and testing datasets were initially loaded into memory as arrays, consuming a considerable amount of RAM. To optimize RAM utilization, an alternative approach involved saving the images to the hard disk and subsequently loading them in batches during training. This strategy effectively mitigated RAM usage, thereby enabling the model to make more efficient use of available memory resources.
- 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] Wevolver. (2023). *Mastering PCB Testing: Techniques, Methods, and Best Practices Unveiled*. Retrieved June 18, 2024, from https://www.wevolver.com/article/pcb-inspection

[2] TensorFlow. (n.d.). *Transfer learning and fine-tuning*. Retrieved June 17, 2024, from https://www.tensorflow.org/tutorials/images/transfer_learning

[3] Ashutosh Shekar. (n.d.). *VGG16*. Retrieved June 17, 2024, from https://github.com/ashushekar/VGG16

[4] Heaton, Jeff. (n.d.). *VGG16 and ImageNet*. Retrieved June 17, 2024, from https://conx.readthedocs.io/en/latest/VGG16%20and%20ImageNet.html

[5] Ultralytics. (n.d.). *YOLOv5*. Retrieved June 20, 2024, from https://github.com/ultralytics/yolov5

[6] Ashraf, H., Waris, M., Ghafoor, M., Gilani, S., & Niazi, I. (2022, March). Melanoma Segmentation using Deep Learning with Test-Time Augmentations and Conditional Random Fields. *Scientific Reports*. Retrieved June 20, 2024, from https://doi.org/10.1038/s41598-022-07885-y

[7] Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., & Le, Q. V. (2020, June). AutoAugment: Learning Augmentation Policies from Data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 113-123). Retrieved June 20, 2024, from https://openaccess.thecvf.com/content_CVPR_2020/html/Cubuk_AutoAugment_Learning_Augmentation_Policies_From_Data_CVPR_2020_paper.html

[8] Albumentations. (n.d.). *Getting Started with Image Augmentation*. Retrieved June 20, 2024, from https://albumentations.ai/docs/getting_started/image_augmentation/

[9] Saponara, S., & Elhanashi, A. E. (2022). Impact of Image Resizing on Deep Learning Detectors for Training Time and Model Performance. In *Applications in Electronics Pervading Industry, Environment and Society* (pp. 10-17). Springer. doi:10.1007/978-3-030-95498-7_2. Retrieved June 20, 2024, from https://doi.org/10.1007/978-3-030-95498-7_2

[10] TensorFlow. (n.d.). *tf.keras.preprocessing.image.ImageDataGenerator*. Retrieved June 20, 2024, from https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

[11] Keras. (n.d.). *Keras API Documentation*. Retrieved June 20, 2024, from https://keras.io/api/

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

[13] 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.

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

[15] 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/.

[16] 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.

[17] Kim, M., Jeong, J., Kim, S. (2021). ECAP-YOLO: Efficient Channel Attention Pyramid YOLO for Small Object Detection in Aerial Image. *Remote Sensing*, **13**(11), 4851. DOI: https://doi.org/10.3390/rs13234851.

[18] Jacob Gildenblat, "Advanced AI Explainability with PyTorch-GradCAM," 2024. [Online]. Available: https://jacobgil.github.io/pytorch-gradcam-book/.

[19] TensorFlow. (n.d.). *tf.keras.losses*. Retrieved June 17, 2024, from https://www.tensorflow.org/api_docs/python/tf/keras/losses

List of Figures