Proceedings of 12th IOE Graduate Conference
Peer Reviewed
ISSN: 2350-8914 (Online), 2350-8906 (Print)
Year: 2022 Month: October Volume: 12

Fault Detection in Printed Circuit Boards using Faster RCNN And YOLOv3

Bipin Thapa Magar [a], Ram Krishna Maharjan [b]

[a, b] *Department of Electronics & Computer Engineering, Pulchowk Campus, IOE, Tribhuvan University, Nepal*
[a] tbipin12@gmail.com, [b] rkmahajn@ioe.edu.np

Abstract
Technology has been an essential part in everyone's daily life. With the development of new technologies, people tend to forget the thing that powers it all i.e. hardware. New and more sophisticated techniques are being developed regularly in the field of hardware design and development. With the step into automation, most large companies tend to print their circuits using automated industries. Although, it increases efficiency, quality assurance is also important. To test and verify the huge number of products, a deep learning object detection system can be developed which can verify the PCB and segregate the faulty ones from the correct ones. This paper focuses on development of object detection networks to detect the various faults in PCB, localize the faults as well as classify them. For this, Faster RCNN network and YOLOv3 have been used and their performance have been measured in terms of Mean Average Precision(mAP). For Faster RCNN, Resnet101 and Resnet50 has been used as the backbone which had mAP of 0.9309 and 0.9578 respectively whereas YOLOv3 performed even better with mAP of 0.991.

Keywords
Faulty PCB, Faster RCNN, YOLO, Object Detection

1. Introduction

With the growth of technology over the world, the race for a better and more efficient hardware has been going on for a long time now. Every hardware manufacturer is opting for a better production environment for their optimized designs. With the increase in the demand, huge production is being done using automated tools and machinery. These machines are prone to small errors. Also, the storage and transport of the circuit boards can be a source of faults and defects. The testing and verification of each of the PCB can be both expensive as well as extensive task to do manually. So, a better approach is to find the faults using computer vision by training a deep learning model.

Normally, human testers monitor the results of more than 50 different tests. The process is both time consuming and costly. The faults are mostly in solder paste comprising of 50%-70% of the total faults [1]. Also, there are mainly two types of test: electrical and non-electrical. The electrical test can find most of the faults but still some may still hide from these tests [2].

In this paper, computer vision based deep learning method has been proposed. Since a single PCB can have multiple faults, the localization of the fault is also necessary. For this, object detection algorithm can be used. Faster RCNN based networks are slower but provide customizability for detection of objects of various sizes whereas starting from YOLOv3, detection of small objects is supported. Since the faults in PCB are relatively small, these two algorithms have been selected for comparison. So, this paper focuses on using object detection algorithm, specifically Faster RCNN and YOLOv3 to find six different faults in the PCB. This paper also compares the performance of the two. Since it is an object detection approach, mAP(mean Average Precision) has been used as the performance measure as it is the standard for object detection algorithms. Two different backbone networks, Resnet101 and Resnet50, have been used in Faster RCNN. So, the three models are trained and their performance based on mAP is compared.

2. Related Works

A lot of research has been done to detect the faults in PCB. Many of them focus on developing image

processing systems whereas some also implement machine learning and deep learning systems.

In 2019, Huang and P. Wei found a scarcity in datasets from previous researches and published their own dataset as open source. They also trained a model based on CNN for the dataset. Their model exhibited a maximum of 99.8% accuracy on their dataset. They used 2 blocks of CNN with block each having 6 layers composed of BN and ReLU function. The proposed method classified the image but didn't localize individual faults in it[3].

Ding et al. published their network named "TDD-net: a tiny defect detection network for printed circuit boards" where they used the dataset from [3] and augmented those data by adding Gaussian noise, changing light, rotating image, flipping, random crop, and shifting and formed 10,668 data with 21,664 faults. They also proposed a network for the defect detection which they named as Tiny Defect Detection Network that uses feature maps on multiple scales. This network is slow as it is based on Faster RCNN and faster implementations such as YOLO was not explored[4].

Reza et al. published "Deep neural network–based detection and verification of microelectronic images" where they used deep neural network to verify the electronic devices in a circuit. They used images from such small devices to train the network and then detect them in the actual circuit. The technique can be translated to detect faults in the circuit as well[5].

Kim et al. took the data from the datasets provided by Huang and P. Wei [3]. Then, they implemented a Convolutional Autoencoder to train the model from the dataset. The autoencoder generates a non defective image which is then subtracted from the original image to find if the input PCB has faults or not. This technique is a different approach as it first tries to fix the faults and then detect it. Although it is a unique approach, it cannot classify the types of faults but gives the locality and shape[6].

Lin et al. published "Feature Pyramid Networks for Object Detection" where they developed a network called Feature Pyramid Network (FPN). The network combined the low-resolution, semantically strong features with high-resolution, semantically weak features to create feature pyramids which are symantically strong at all levels[7].

Chaudhary et al. published "Automatic visual inspection of printed circuitboard for defect detection and classification" where they detect all 14 types of defects as at fast as 2.528 seconds. The paper proposes converting the initial image of PCB to greyscale and then applying median and low pass filtering with Gaussian Filter. They then segment the tracks, holes and soldering pads which they then use to find the difference with the non faulty image. They filter out and eliminate the small areas and then detect the final difference which they can then classify. The classification is done by separating the type of image formed after the above mentioned process based on an algorithm mentioned in the paper[8].

Nayak et al. proposed a simple method to detect the faults in PCB by using image subtraction method. At first the image is taken which is then corrected for lighting problems. It is then checked for tilt using Hough Transform. The PCB border is then captured and the image is compared with the template image. This is relatively simple method but still the class of fault is not correctly recognized in this method[9].

Guohua Liu and Haitao Wena published "Printed circuit board defect detection based on MobileNet-Yolo-Fast" where they used the same dataset as [4] and used YOLO for the fault detection. They implemented Mobilenet-YOLO-fast for the fault detection. For the anchors, they implemented K-means clustering to define reasonable anchor size[10].

Tan et al. published "Comparison of YOLO v3, Faster R-CNN, and SSD for Real-Time Pill Identification" where they used the three object detection algorithms on pill image dataset and compared the developed models. They found that the Faster RCNN has a higher mAP than YOLO v3 but in terms of speed, YOLO performed almost 8 times faster. SSD was slower as well as had less mAP than both the algorithms. This comparison was done for Pill dataset and that of PCB has not been done[11].

3. Proposed Methodology

The proposed methodology comprises of various sections as shown in Figure 1.

3.1 Image Acquisition

For the image acquisition, we take the image of PCB from camera. For this, we first mount the camera over the PCB and supply sufficient lighting.

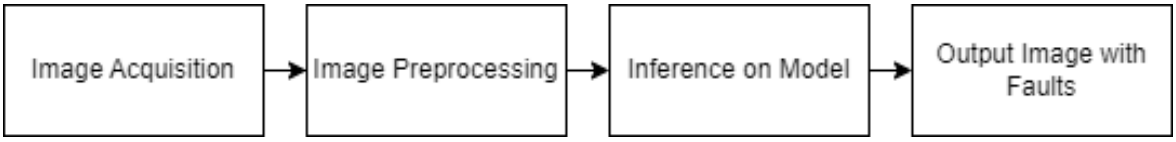

Figure 1: Overall Block Diagram

3.2 Image Preprocessing

The image is then sent through a series of steps mentioned in Algorithm 1.

Algorithm 1 Pre-processing

1: Load Image to *img*
2: $blurred \leftarrow GaussianBlur(img, filter = 7x7)$
3: $threshold \leftarrow AdaptiveThreshold(blurred, max = 255, threshold = Gaussian, blocksize = 27, c = 6)$
4: $postBlur \leftarrow GaussianBlur(threshold, filter = 19x19)$
5: $erode \leftarrow Erosion(postBlur, filter = 7x7)$
6: Find contours from *erode* and save to *contours*
7: $largest \leftarrow max(contours, key = area)$ which is the PCB
8: $x, y, w, h \leftarrow BoundingRectangle(largest)$
9: Crop the original image to x, y, w, h to *image*

3.3 Faster RCNN

Faster RCNN algorithm[12] has been implemented in the research to detect the various faults in the PCB. With the improvement over RCNN[13] and Fast RCNN[14], Faster RCNN provides a single stage network as well as introduces Region Proposal Network(RPN). Faster RCNN has the advantage of higher mean Average Precision(mAP) with reduced frames per second(FPS) as compare to other algorithms such as You Only Look Once(YOLO) or Single Shot Detectors(SSD)[11].

The input to the system is the augmented 600 x 600 image of the PCB. The backbone is first setup as Resnet101 as the main Convolutional Network. We then modify it to Resnet50 to compare the performance. Also, multi-scale feature fusion strategy is adopted which is discussed in detail in the later section. The ROI pooling layer is followed by 2 fully connected layers which then finally splits to classification and Bounding box regressor. Similary, the system architecture with Resnet50 as backbone is shown in Figure 3.

3.3.1 Resnet as the backbone

Resnet101[15] has been used in this paper mentioned in [4]. Here also, at first Resnet101 has been used as the backbone Convolutional Network to extract features. The Resnet101 used has been pretrained on the ImageNet classification set. We then tune the network using our own training set. The ResNets are provided with five residual blocks $\{conv2\_x, conv3\_x, conv4\_x, conv5\_x\}$. These residual blocks are selected and their outputs are denoted as $\{$C2, C3, C4, C5$\}$ respectively. We use the strides of $\{4, 8, 16, 32\}$ pixels respectively for the residual blocks with respect to input image.

3.3.2 Multilayer Fusion

By combining structurally strong high-resolution feature maps and semantically strong low-resolution feature maps, we can detect the faults which can be classified as low-level vision task. So, the high level and low level feature maps are contatenated to form $\{$P2, P3, P4, P5$\}$ from $\{$C2, C3, C4, C5$\}$[7].
First, we get the P_5 layer using a convolutional layer as shown below:

$$P_5 = Conv2D(C_5, outputs = 256, kernel = [1,1], stride = 1) \quad (1)$$

Then, P_6 is calculated by max pooling P_5 with the following:

$$P_6 = MaxPool2D(P_5, kernel = [1,1], stride = 2) \quad (2)$$

The remaining P_2 to P_4 is created by fusing two layers as shown in the system architecture in Figure 2 and Figure 3. For each of the P_i, the fusion is done as:

$$P_i = fusion(C_i, P_{i+1}) \quad (3)$$

Using the above equation, we calculate P_2 to P_4. For fusion, we have three steps. First, we upsample the P_{i+1} with the size of C_i as:

$$upsample_i = bilinear\_resize(P_{i+1} to sizeof C_i) \quad (4)$$

Then, we reduce the output channels of the convolutional layer to 256 using a 1x1 convolutional

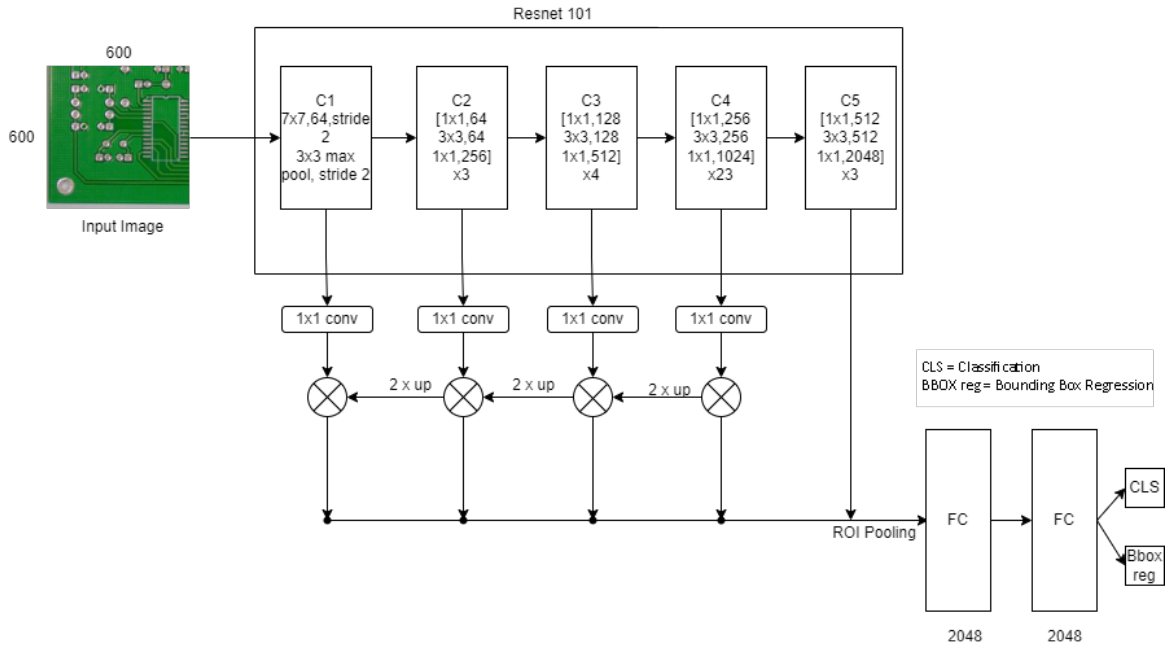

Figure 2: System Architecture with Resnet101 as backbone

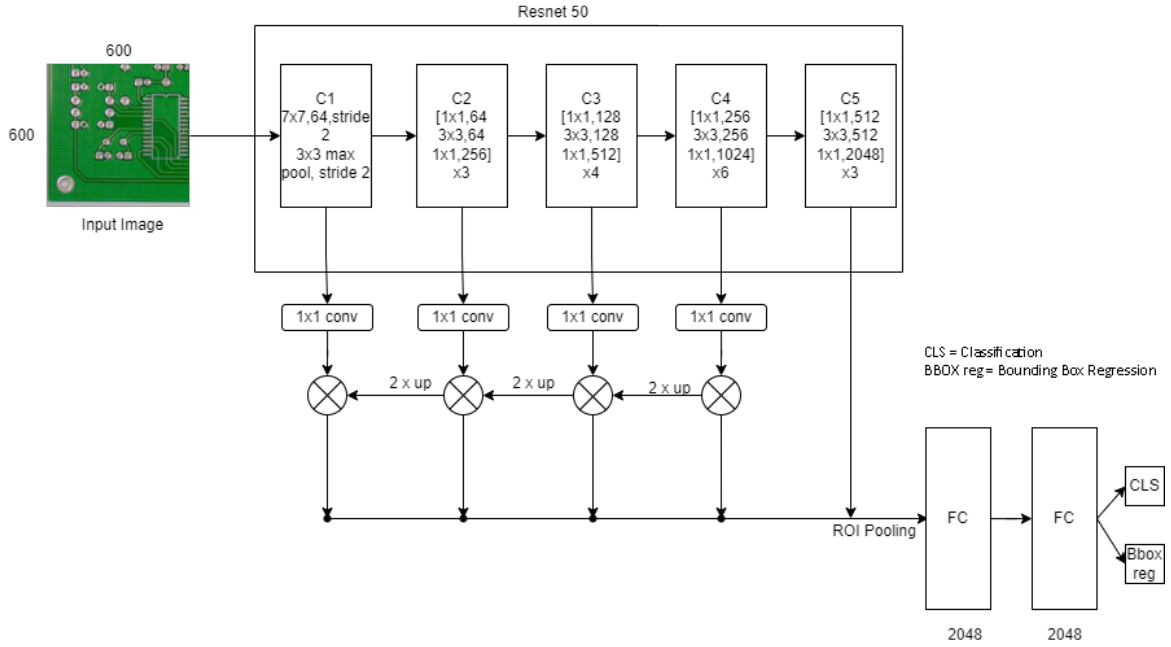

Figure 3: System Architecture with Resnet50 as backbone

layer as:

$$reduced\_dim_i = Conv2D(C_i, outputs = 256, kernel = [1,1], stride = 1) \quad (5)$$

We finally fuse this to get the new fused feature as:

$$P_i = fusion(C_i, P_{i+1}) = \frac{1}{2}(upsample_i + reduced\_dim_i) \quad (6)$$

Each of these fused feature from P_2 to P_4 is finally convoluted as follows:

$$P_i = Conv2D(P_i, outputs = 256, kernel = [3,3], stride = 1) \quad (7)$$

3.3.3 Anchors

The researchers of Tiny Defect Detection [4] used K means clustering on the dataset to find the optimum anchor sets as the defects in PCB is small relative to the image and traditional anchor size is not suitable. We are using the same anchor set which given in Table 1.

Table 1: Anchor Generation for each Layer

Layer	Anchor Size	Stride
P_2	15x15	4
P_3	25x25	8
P_4	40x40	16
P_5	60x60	32
P_6	80x80	64
Scales	2, 3, 4	
Ratios	2, 3, 4, 5	

3.3.4 Postprocess the region proposals

This step has 4 major steps:

1. Decode Boxes: In this step, we take in the respective anchor parameters and decode the box to generate respective proposals.

2. Clip to image boundaries: In this step, we clip the bounding boxes decoded to the actual image boundaries.

3. Select top N: Since, there can be a large number of region, we only take 12,000 top region for Non-Max Suppression. For this, we select the top 12,000 based on the highest class probability calculated. We choose from a maximum of 12,000 regions but if we have less than that, we select from those. For testing, we take a total of 6000 regions.

4. Non-Max Suppression: Now, we take the region proposals and apply non-max suppression. During training, we select 2000 from the 12,000 regions and during testing, we select 1000 from the 6000 regions. For this, we use the IoU threshold of 0.7.

3.3.5 Region of Interest(ROI) Pooling

In this step, we take each of the ROI from the proposed regions and apply two steps. We first crop each of the image of the feature map based on the ROI with the size of 14x14. Then, We apply max pooling on the cropped ROI as:

$$Pooled\_features = MaxPool2D(Cropped, kernel = [2,2], stride = 2) \quad (8)$$

3.3.6 Fully Connected Layer and Prediction

Finally, we flatten the pooled features from the previous section as:

$$flatten = Flatten(Pooled\_features) \quad (9)$$

Then, we add two fully connected layers as :

$$FC_1 = fully\_connected(flatten, outputs = 2048, activation = softmax) \quad (10)$$

$$FC_2 = fully\_connected(FC_1, outputs = 2048, activation = softmax) \quad (11)$$

Finally, we take the output from FC_2 and pass it to two fully connected layers: one for class prediction and another for bbox prediction.

$$class\_pred = fully\_connected(FC_2, outputs = 7, activation = None) \quad (12)$$

$$bbox\_pred = fully\_connected(FC_2, outputs = 28, activation = None) \quad (13)$$

3.3.7 Loss Functions

For loss, the total loss is calculated by taking a weighted sum of four different losses of rpn_loss_bbox, rpn_loss_cls, fast_rcnn_loss_bbox and fast_rcnn_loss_cls. The classification loss L_{cls} is calculated using the following:

$$L_{cls}(p_i, p_i^*) = -log(p_i^* p_i + (1-p_i^*)(1-p_i)) \quad (14)$$

Here, p_i is predicted probability of anchor i being an object and p_i^* is the ground truth label i.e. p_i^* = 1 for positive label and 0 for negative label.
The regression loss is then given as:

$$L_{reg}(t_i, t_i^*) = R(t_i, t_i^*) \quad (15)$$

Here, R represents Smooth L1 function. Also, t_i is the vector representing the 4 parameterized coordinates of the predicted bounding box and t_i^* is a ground truth box associated with a positive anchor.
The various loss curves in Fast RCNN as well as RPN are shown in Figure 4.

3.4 YOLO (You Only Look Once)

YOLOv3 has been implemented to detect the faults in the PCB. This is because YOLO[16] and YOLOv2 (YOLO9000)[17] didnot perform well on small objects and PCB faults are generally very small. For this, the architecture of YOLOv3 is shown in Figure 5. Darknet-53[18] has been followed by the detection network which has three prediction layers which gives output in 3 scales with stride of 8, 16 and 32. The details about the anchors has been discussed later on this chapter. Also, the input to the network has been set to 416 by first cropping and converting all training and test dataset to that size. It is done because YOLO needs input of size divisible by 32[19].

3.4.1 Hyperparameters

The hyperparameters used during the training is given in Table 2.

3.4.2 Anchors

YOLO uses K-means clustering to define anchors based on the dataset. For 416x416 input, K-means clustering was applied to get 9 anchors as shown in Table 3.

3.4.3 Loss Functions

YOLO uses three different loss functions: classification loss, localization loss and confidence(objectness) loss. The classification loss at each cell is given in Equation 16.

Table 2: Training hyperparameters

Parameter	Value
momentum	0.86714
weight_decay	0.00058
epochs	300
batch_size	8
imgsz	416
workers	8
optimizer	SGD

Table 3: YOLOv3 anchors

Layer	Anchor Size	Stride
Predict1	12x12 15x16 22x13	8
Predict2	13x25 20x19 32x16	16
Predict3	18x33 25x25 33x31	32

$$Loss_{class} = \sum_{i=0}^{S^2} 1_i^{obj} * \sum_{c \varepsilon classes} (p_i(c) - \hat{p}_i(c))^2 \quad (16)$$

where,
1_i^{obj} = 1 if object is in cell i, else 0,
$\hat{p}_i(c)$ is conditional class probability for class c in cell i,
The localization loss is error in the predicted box, location and size as shown in Equation 17.

$$Loss_{loc} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \quad (17)$$

where,
1_{ij}^{obj} = 1 if j^th bbox in cell i is responsible for detecting object, else 0,
λ_{coord} increases weight for loss in bbox coordinates i,
The confidence loss measures the objectness of a box

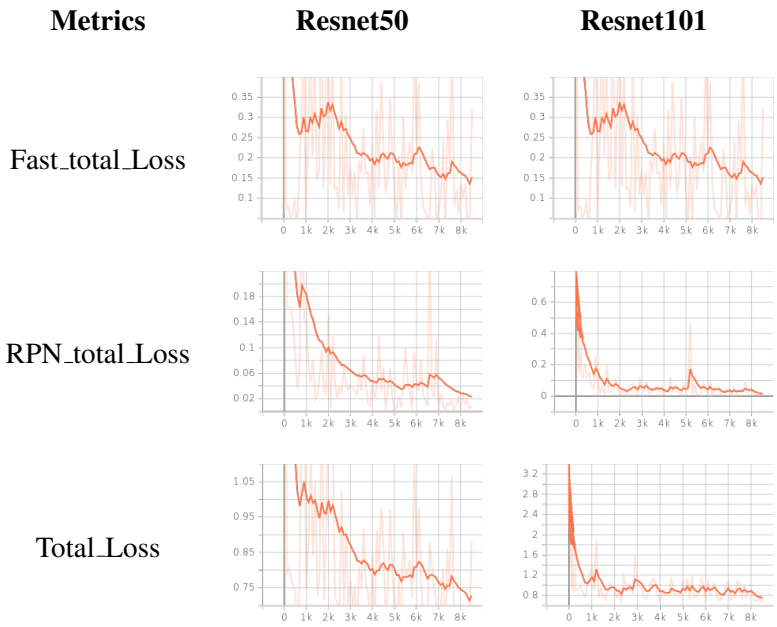

Figure 4: Loss

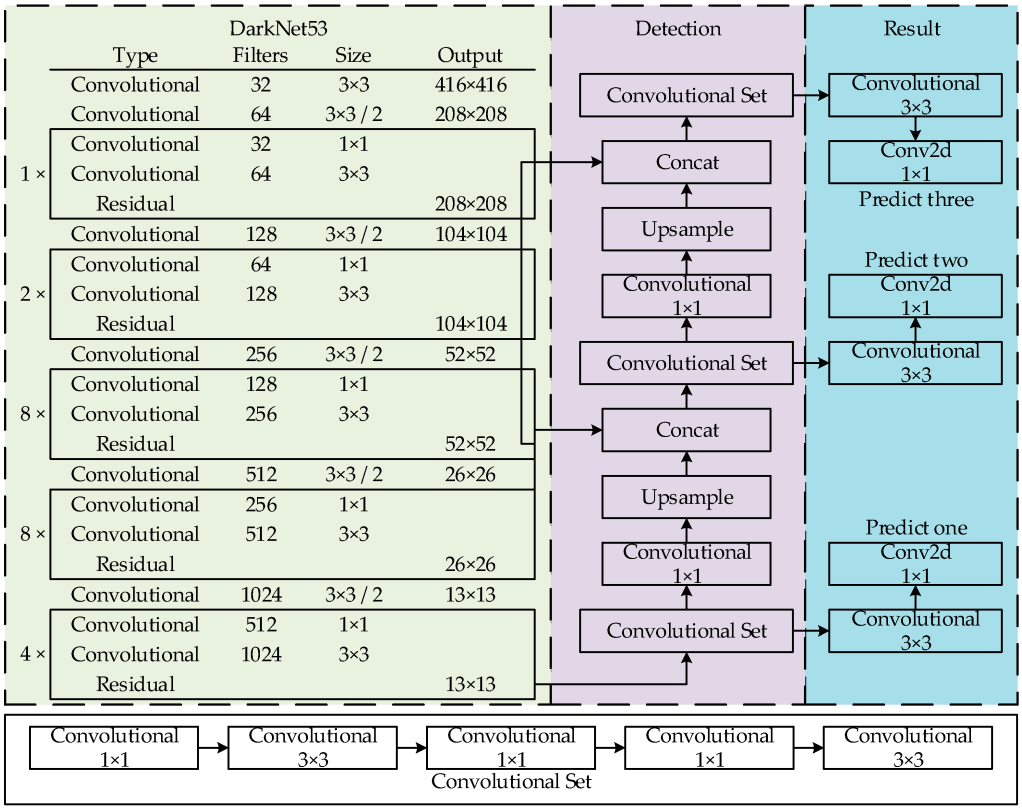

Figure 5: YOLOv3 Architecture [20]

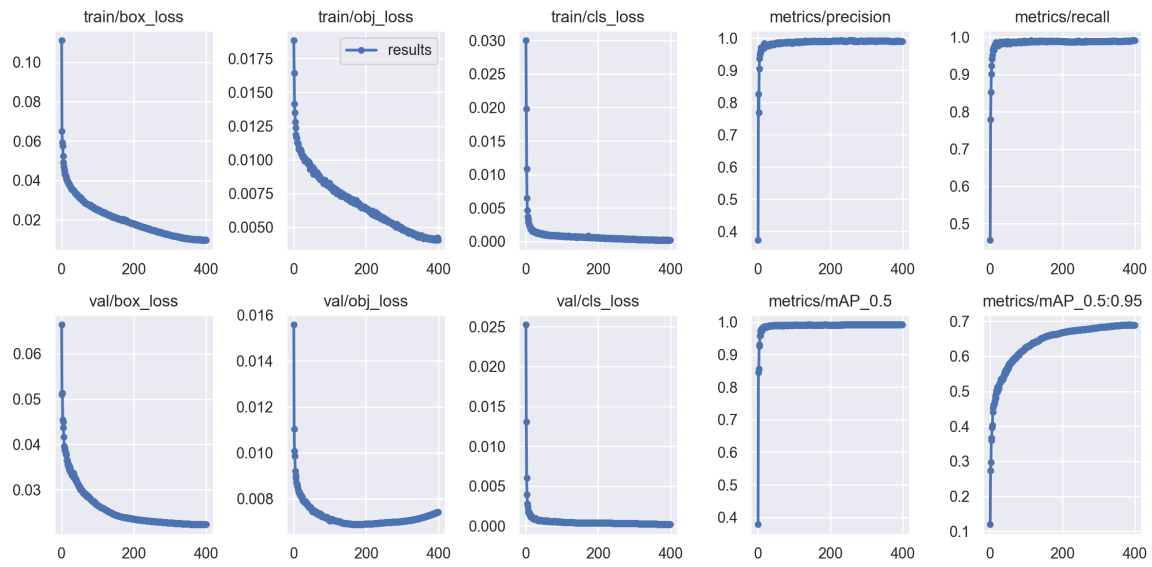

Figure 6: YOLO training results

as shown in Equation 18.

$$Loss_{obj} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 \tag{18}$$

If no object is detected, the confidence loss is given in Equation 19.

$$Loss_{obj} = \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 \tag{19}$$

where,
1_{ij}^{obj} = 1 if j^th bbox in cell i is responsible for detecting object, else 0,
$\hat{C}_i$ is box confidence score f box j in cell i,
λ_{noobj} = 1 if object is detected, else it weighs down loss when detecting background
So, the total loss is the sum of the three given by Equation 20.

$$\begin{aligned} Loss_{total} &= \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\ &+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \\ &+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 \\ &+ \sum_{i=0}^{S^2} 1_i^{obj} * \sum_{c \varepsilon classes} (p_i(c) - \hat{p}_i(c))^2 \end{aligned} \tag{20}$$

The loss curves in training is shown in Figure 6.

4. Results and Analysis

The training accuracy curve for Resnet50 as well as Resnet101 is shown in Figure 7. For Resnet101, the Fast RCNN accuracy is upto 99.62% and RPN accuracy is upto 99.97% at the end of the training. For Resnet50, the Fast RCNN accuracy is upto 96.15% and RPN accuracy is upto 99.72% at the end of the training.

The Precision-Recall graph using the Resnet101 network is shown in Figure 8. The mean average precision is 0.9309.

The Precision-Recall graph using the Resnet50 network is shown in Figure 9. The mean average precision is 0.9578.

The Precision-Recall graph using the YOLOv3 network is shown in Figure 10. The mean average precision is 0.991.

Although Faster RCNN based algorithms have a relatively high mAP, YOLOv3 has even greater mAP for the given PCB dataset. So, among the two algorithms, YOLOv3 performed better than Faster RCNN. The detailed AP is given in Table 4.

An example of an input of the test data is shown in Figure 11. The output from the Resnet101 based Faster RCNN is shown in Figure 12 whereas that of Resnet50 is shown in Figure 13. Also, the output from YOLOv3 is shown in Figure 14.

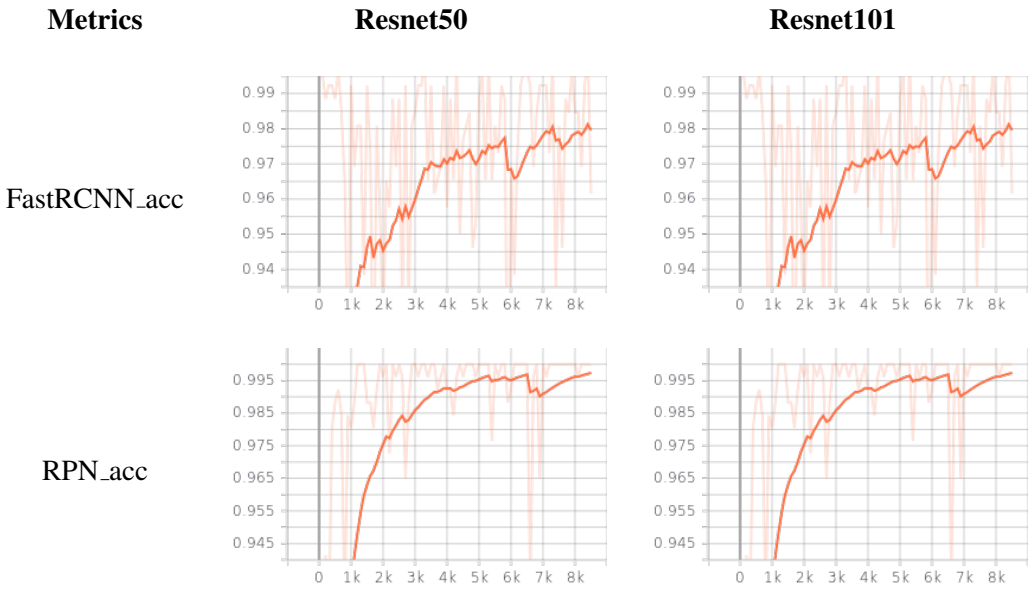

Figure 7: Training Accuracy Curve

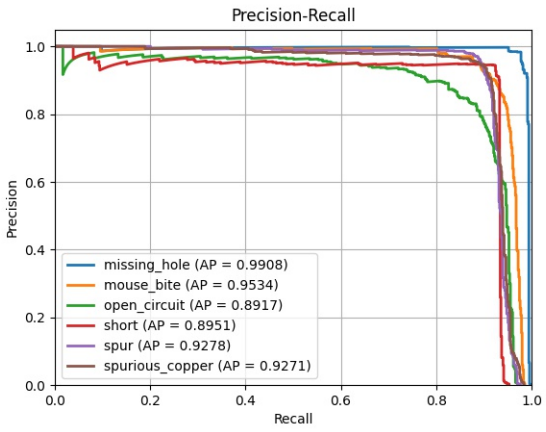

Figure 8: Precision vs Recall (Resnet101)

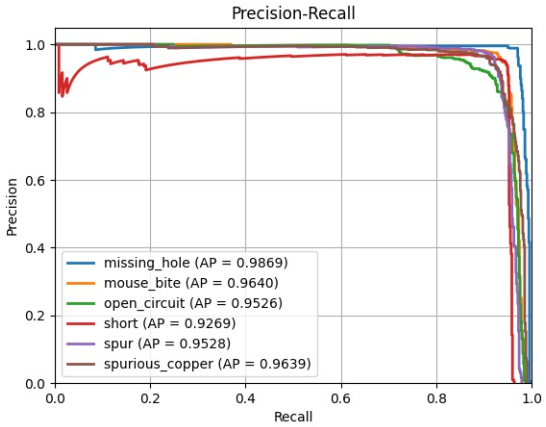

Figure 9: Precision vs Recall (Resnet50)

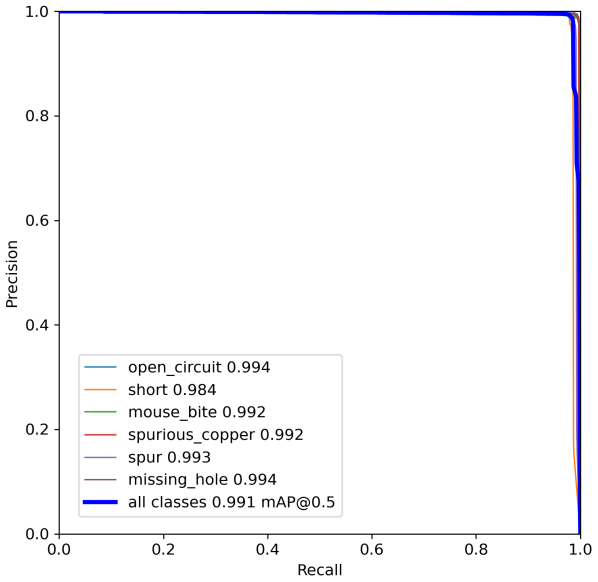

Figure 10: Precision vs Recall (YOLOv3)

Table 4: Average Precision

Network	Resnet101	Resnet50	YOLOv3
Missing Hole	.9908	.9869	.994
Mouse Bite	.9534	.9640	.992
Open	.8917	.9526	.994
Short	.8951	.9269	.984
Spur	.9278	.9528	.993
Spurious Copper	.9271	.9639	.992
mAP	**.9309**	**.9578**	**.991**

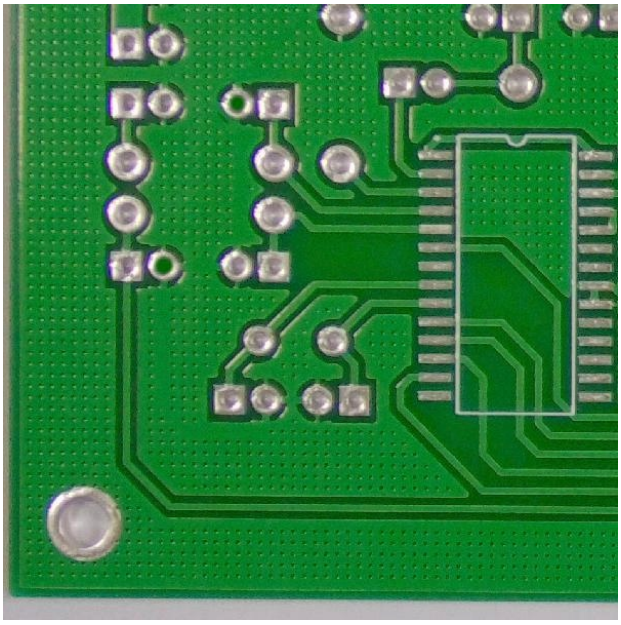

Figure 11: Test Input PCB

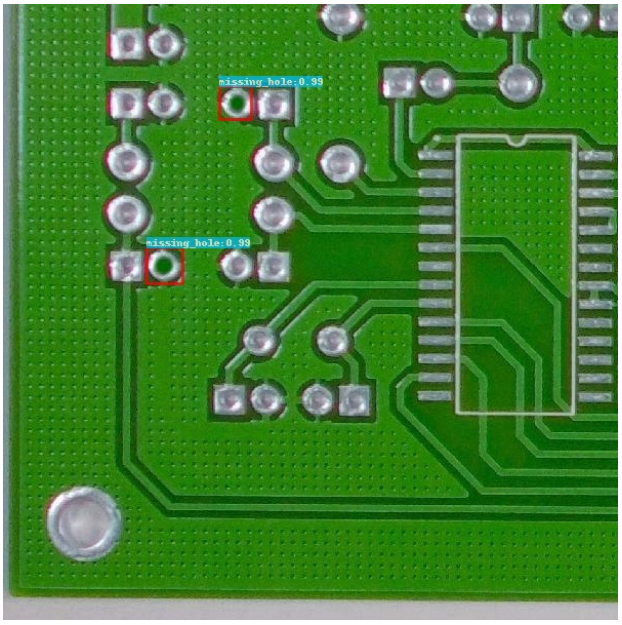

Figure 13: Output (Resnet50)

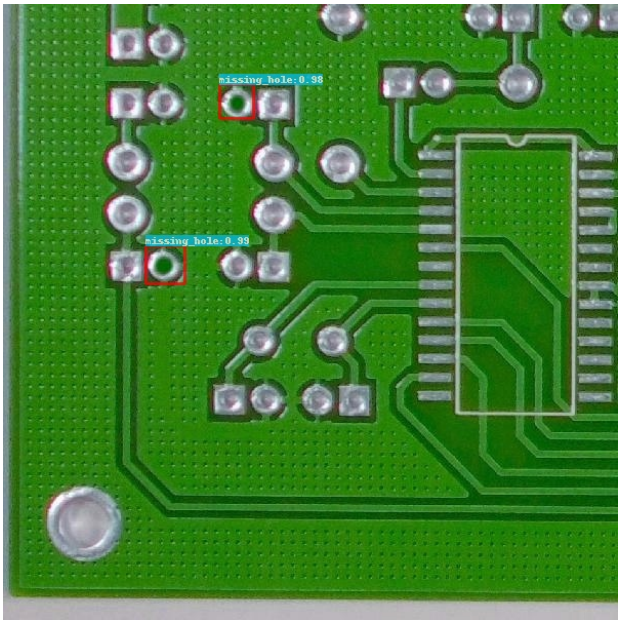

Figure 12: Output (Resnet101)

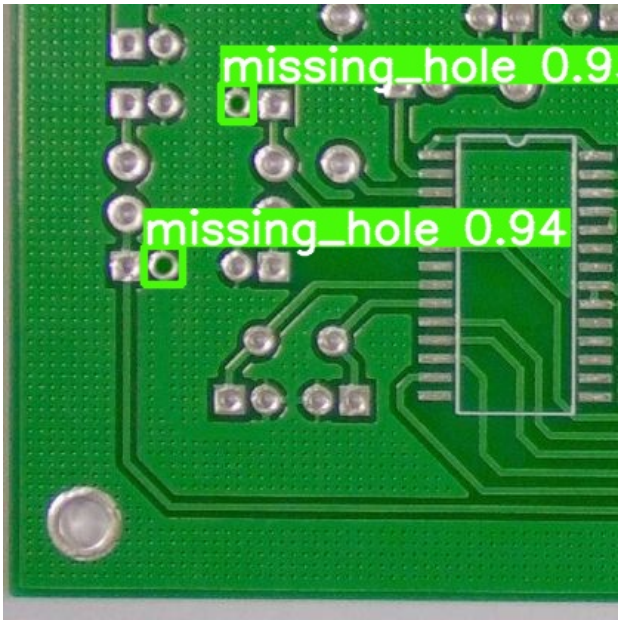

Figure 14: Output (YOLOv3)

Acknowledgements

This research was supported by Department of Electronics and Computer Engineering, Pulchowk Campus, Tribhuvan University and all the respected faculties under the department. A special thanks to Dr. Surendra Shrestha, PHD, for his insightful suggestions during the entirety of the research.

References

[1] S S Zakaria, A Amir, N Yaakob, and S Nazemi. Automated detection of printed circuit boards (PCB) defects by using machine learning in electronic manufacturing: Current approaches. *IOP Conference Series: Materials Science and Engineering*, 767(1):012064, feb 2020.

[2] Madhav Moganti, Fikret Ercal, Cihan H. Dagli, and Shou Tsunekawa. Automatic pcb inspection algorithms: A survey. *Computer Vision and Image Understanding*, 63(2):287–313, 1996.

[3] Weibo Huang and Peng Wei. A pcb dataset for defects detection and classification. *arXiv preprint arXiv:1901.08204*, 2019.

[4] Runwei Ding, Linhui Dai, Guangpeng Li, and Hong Liu. Tdd-net: a tiny defect detection network for printed circuit boards. *CAAI Transactions on Intelligence Technology*, 4(2):110–116, 2019.

[5] Md Alimoor Reza, Zhenhua Chen, and David J Crandall. Deep neural network–based detection and verification of microelectronic images. *Journal of Hardware and Systems Security*, 4(1):44–54, 2020.

[6] Jungsuk Kim, Jungbeom Ko, Hojong Choi, and Hyunchul Kim. Printed circuit board defect detection using deep learning via a skip-connected convolutional autoencoder. *Sensors*, 21(15), 2021.

[7] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[8] Vikas Chaudhary, Ishan R. Dave, and Kishor P. Upla. Automatic visual inspection of printed circuit board for defect detection and classification. pages 732–737, 2017.

[9] Jithendra P R Nayak, K Anitha, B D Parameshachari, Reshma Banu, and P Rashmi. PCB fault detection using image processing. *IOP Conference Series: Materials Science and Engineering*, 225:012244, aug 2017.

[10] Guohua Liu and Haitao Wen. Printed circuit board defect detection based on mobilenet-yolo-fast. *Journal of Electronic Imaging*, 30(4):043004, 2021.

[11] Lu Tan, Tianran Huangfu, Liyao Wu, and Wenying Chen. Comparison of yolo v3, faster r-cnn, and ssd for real-time pill identification. 2021.

[12] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[13] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[14] Ross Girshick and RCNN Fast. Microsoft research. *Fast r-cnn*, 27, 2015.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[17] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[18] Joseph Redmon. Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`, 2013–2016.

[19] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[20] Haipeng Zhao, Yang Zhou, Long Zhang, Yangzhao Peng, Xiaofei Hu, Haojie Peng, and Xinyue Cai. Mixed yolov3-lite: A lightweight real-time object detection method. *Sensors*, 20(7), 2020.