Vision System For PCB Fault Detection

A project report submitted in partial fulfillment of
the requirements for the degree of

Bachelor of Technology

by

Anuja Gundecha	161298
Rohit Kulkarni	161248
Harshit Shukla	161551

Under the guidance of
Prof. Rupali Tornekar

DEPARTMENT OF ELECTRONICS ENGINEERING
VISHWAKARMA INSTITUTE OF TECHNOLOGY PUNE

2019 - 2020

Bansilal Ramnath Agarwal Charitable Trust's

VISHWAKARMA INSTITUTE OF TECHNOLOGY, PUNE - 37

(An Autonomous Institute Affiliated to Savitribai Phule Pune University)

CERTIFICATE

This is to certify that the project report entitled Vision System For PCB Fault Detection has been submitted in the academic year 2019-20 by

Anuja Gundecha	(GR No. 161298)
Rohit Kulkarni	(GR No. 161248)
Harshit Shukla	(GR No. 161551)

under the supervision of Prof. Rupali Tornekar in partial fulfillment of the requirements for the degree of Bachelor of Technology in Electronics Engineering.

Guide/Supervisor
Prof. Rupali Tornekar
Signature:

Head of the Department
Prof. (Dr.) V. D. Gaikwad
Signature:

External Examiner
Name:
Signature:

Acknowledgments

We are thankful to all the people who are directly or indirectly responsible for this project to become successful. Firstly, we would like to thank Prof. Dr. Vijay Gaikwad, Head of Electronics Department for encouraging us and providing us with this opportunity and various facilities for completion of our project.

We pay our deep sense of gratitude to our project guide Prof. Rupali Tornekar for her guidance and support starting right from literature survey of the project to the final implementation and interpretation of results. We are immensely obliged to her for her elevating inspiration, encouraging guidance and kind supervision in the completion of our project. We would like to thank Amit Nahar sir from NSYS systems for their guidance and support. Lastly, we would also thank the Analytics Vidhya for their database that had images annotations of images.

Date: _____

Anuja Gundecha

Rohit Kulkarni

Harshit Shukla

Contents

List of Tables

List of Figures

Abstract

A printed circuit board (PCB) mechanically supports and electrically connects electronic components or electrical components using conductive tracks, pads and other features etched from one or more sheet layers of copper laminated onto and/or between sheet layers of a non-conductive substrate. Components are generally soldered onto the PCB to both electrically connect and mechanically fasten them to it.Printed circuit boards are used in all but the simplest electronic products. They are also used in some electrical products, such as passive switch boxes.PCBs can be single-sided (one copper layer), double-sided (two copper layers on both sides of one substrate layer), or multi-layer (outer and inner layers of copper, alternating with layers of substrate). Multi-layer PCBs allow for much higher component density, because circuit traces on the inner layers would otherwise take up surface space between components. The rise in popularity of multilayer PCBs with more than two, and especially with more than four, copper planes was concurrent with the adoption of surface mount technology. However, multilayer PCBs make repair, analysis, and field modification of circuits much more difficult and usually impractical.

Printed circuit board (PCB) is the fundamental carrier in electronic devices on which a great number of elements are placed. To avoid the shortcoming of manual detection, easily being fatigued, low efficiency, for instance, automated optical inspection (AOI) based on machine vision has been widely used in industry. As PCB becomes more and more complicated, the tasks of detection and classification defects are also more difficult than before. Hence this report focusses on formulating a method which will detect the faults in PCB with greater accuracy and efficiency. The method goes around one of the hottest topics in the present market regarding Convolutional Neural Networks (CNN) and its types. Thus, this project will explain how complex neural networks can help us building a model that can detect the faults in PCB with greater accuracy and precision.

Chapter 1

Literature Survey

The importance of the Printed Circuit Board inspection process has been magnified by requirements of the modern manufacturing environment where delivery of 100 percent defect free PCBs is the expectation. To meet such expectations, identifying various defects and their types becomes the first step. In this PCB inspection system the inspection algorithm mainly focuses on the defect detection using the natural images. Many practical issues like tilt of the images, bad light conditions, height at which images are taken etc. are to be considered to ensure good quality of the image which can then be used for defect detection. Printed circuit board (PCB) fabrication is a multidisciplinary process, and etching is the most critical part in the PCB manufacturing process. The main objective of Etching process is to remove the exposed unwanted copper other than the required circuit pattern. In order to minimize scrap caused by the wrongly etched PCB panel, inspection has to be done in early stage. However, all of the inspections are done after the etching process where any defective PCB found is no longer useful and is simply thrown away. Since etching process costs 0 percent of the entire PCB fabrication, it is uneconomical to simply discard the defective PCBs. In this paper a method to identify the defects in natural PCB images and associated practical issues are addressed using Software tools and some of the major types of single layer PCB defects are Pattern Cut, Pin hole, Pattern Short, Nick etc., Therefore the defects should be identified before the etching process so that the PCB would be reprocessed. In the present approach expected to improve the efficiency of the system in detecting the defects even in low quality images.

Bare printed circuit board is a PCB without any placement of electronic components . In order to reduce cost spent on manufacturing caused by defected bare PCB,

the bare PCB must be check firstly.Printed circuit board (PCB) is a mechanical platform of conductive and non-conductive layers on which electronic components are electrically connected. Glass fibers and epoxy is used to form non-conductive layer of the PCB and copper, nickel, aluminum, chrome and other certain metals are used form the conductive layer of the PCB. Fabrication of PCB is a multilevel process inhibiting several multidisciplinary processes. Etching is one such process in the PCB fabrication wherein unwanted metallic portion occurring in places other than the designed pattern is removed. If the PCB defects are still present in the final etching inspection process ,PCBs are considered as the scrap and can be thrown out from process .The 70 percent of the PCB fabrication cost is utilized for etching process ,so we mainly focus on ethching process. We detect fault and classify them based on overetch ,underetch or both. We using subtraction method to detect fault by comparing template PCB and real time PCB

Printed circuit board (PCB) is the fundamental carrier in electronic devices on which a great number of elements are placed. The quality of the PCB will directly impact the performance of electronic devices. To avoid the shortcoming of manual detection, easily being fatigued, low efficiency, for instance, automated optical inspection (AOI) based on machine vision has been widely used in industry. As PCB becomes more and more complicated, the tasks of detection and classification defects are also more difficult than before. Currently, there are few public datasets on the Internet on PCB, many methods proposed in published papers used their own images, which is not convenient for other researchers to compare their new methods. For the purpose of solving above problems, we produced a public colorized synthesized PCB dataset with defects that is available to other people who want to design and evaluate their approaches. Conventional AOI methods for inspecting printed circuit board can be divided into 3 main streams, reference comparison approach, non-reference verification approach, and hybrid approach. In reference comparison approach, a standard image which is called template will be prepared firstly, and then a PCB need to be inspected will be compared with the template to find the unknown defects. Though it is straightforward and easy to use, there are also many factors that we have to take in consideration, unbalanced illumination, inaccurate registration, vast storage requirements, etc. In the non-reference verification approach, the aim of the method is to find out if wiring track, pad and hole are in the compliance with design without a template board. This approach does not have the limits of reference method,

nevertheless, it may have difficulties in detecting large defects. In the hybrid approach, reference method and non-reference method are combined, this approach will have the merits of the two basic methods, meanwhile, it requires high computation capacity.

Due to the rapid development of printed circuit board (PCB) design technology, inspection of PCB surface defects has become an increasingly critical issue. The classification of PCB defects facilitates the root causes of detects' identification. As PCB defects may be intensive, the actual PCB classification should not be considered as a binary or multi-category problem. This type of problem is called multilabel classification problem. Recently, as one of the deep learning frameworks, convolutional neural network (CNN) has a major breakthrough in many areas of image processing, especially in the image classification. This paper proposes a multi-task convolutional neural network model to handle the multi-label learning problem by defining each label learning as a binary classification task. In this paper, the multi-label learning is transformed into multiple binary classification tasks by customizing the loss function. Extensive experiments demonstrate that the proposed method achieves great performance on the dataset of Defects. A variety of approaches for automated visual inspection of printed circuits have been reported over the past two decades. Algorithms and techniques for the automated inspection of printed circuit boards are examined. A classification tree for these algorithms is presented and the algorithms are grouped according to this classification. The method proposed earlier concentrates mainly on image analysis and fault detection strategies; these also include state-of-the-art techniques. A summary of the commercial PCB inspection systems is also presented.

Printed Circuit Board (PCB) inspection is an essential part of PCB production process. Traditional PCB bare board defect detection methods have their own defects. However, the PCB bare board defect detection method based on Automatic Optic Inspection (AOI) is a feasible and effective method, and it is having more and more application in industry. Based on the idea of the reference comparison method, this paper aims at studying the classification of defects. First of all, the method of extracting defect areas using morphology is studied, meanwhile, a data set containing 1818 images with 6 different detailed defect area image parts are produced. Then, in order to classify defects accurately, a traditional classification algorithm based on digital image processing was attempted, and a defect classification algorithm based on convolutional neural network

(CNN) was proposed. After experimental demonstration, in the actual results, the defect classification algorithm based on convolutional neural network can achieve a fairly high classification accuracy (95.7 percent), which is much higher than the traditional method, and the new method has stronger stability than the traditional one. Now in this proposed methodology, detection of fault PCB images is done with the help of convolutional neural networks (CNN). So let's go through the basic building blocks of the proposed methodology:

Chapter 2

Convolutional Neural Networks (CNN)

2.1 What is CNN?

In neural networks, Convolutional neural network (CNNs) is one of the main categories to do images recognition, images classifications. Objects detections, recognition faces etc., are some of the areas where CNNs are widely used.CNN image classifications take an input image, process it and classify it under certain categories (Example: Dog, Cat, Tiger, Lion). Computers sees an input image as array of pixels and it depends on the image resolution. Based on the image resolution, it will see h x w x d(h = Height, w = Width, d = Dimension). Example: An image of 6 x 6 x 3 array of matrix of RGB (3 refers to RGB values) and an image of 4 x 4 x 1 array of matrix of grayscale image.Technically, deep learning CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernels), Pooling, fully connected layers (FC) and apply SoftMax function to classify an object with probabilistic values between 0 and 1. The below figure is a complete flow of CNN to process an input image and classifies the objects based on values.

2.1.1 What is convolutional layer?

Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel.

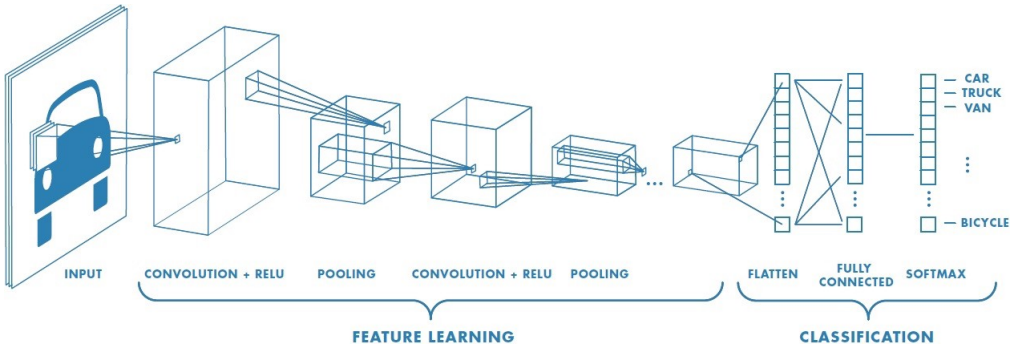

Figure 2.1: Neural network with many convolutional layers

- An image matrix (volume) of dimension **(h x w x d)**
- A filter **(f_h x f_w x d)**
- Outputs a volume dimension **(h - f_h + 1) x (w - f_w + 1) x 1**

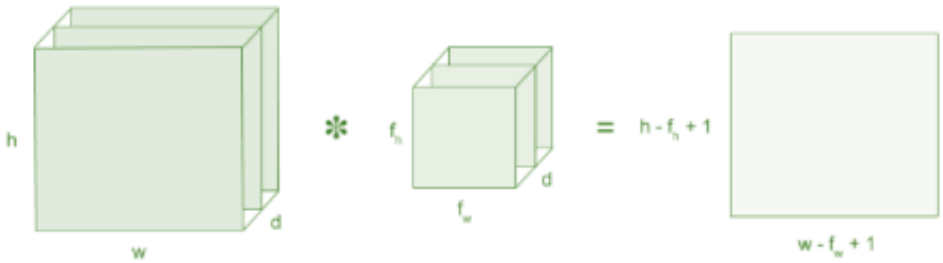

Figure 2.2: Image matrix multiplies kernel or filter matrix

Consider a 5 x 5 whose image pixel values are 0, 1 and filter matrix 3 x 3 as shown in below.

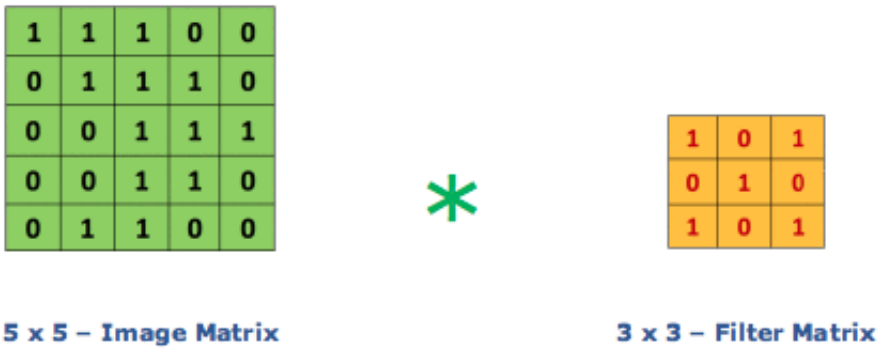

Figure 2.3: Image matrix multiplies kernel or filter matrix

Then the convolution of 5 x 5 image matrix multiplies with 3 x 3 filter matrix which is called “Feature Map” as output. Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters.

2.1.2 What are Strides?

Stride is the number of pixels shifts over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on. The below figure shows convolution would work with a stride of 2.

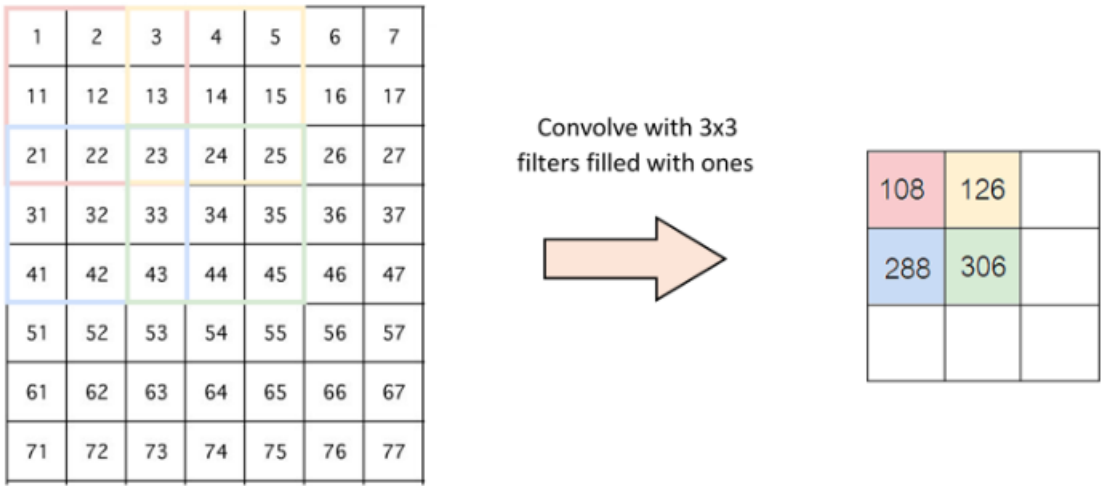

Figure 2.4: Stride of 2 pixels

2.1.3 What is padding?

Sometimes filter does not fit perfectly fit the input image. We have two options:

1. Pad the picture with zeros (zero-padding) so that it fits
2. Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.

2.1.4 What is pooling?

Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called subsampling or downsampling which reduces the di-

mensionality of each map but retains important information. Spatial pooling can be of different types:

1. Max Pooling
2. Average Pooling
3. Sum Pooling Max pooling takes the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.

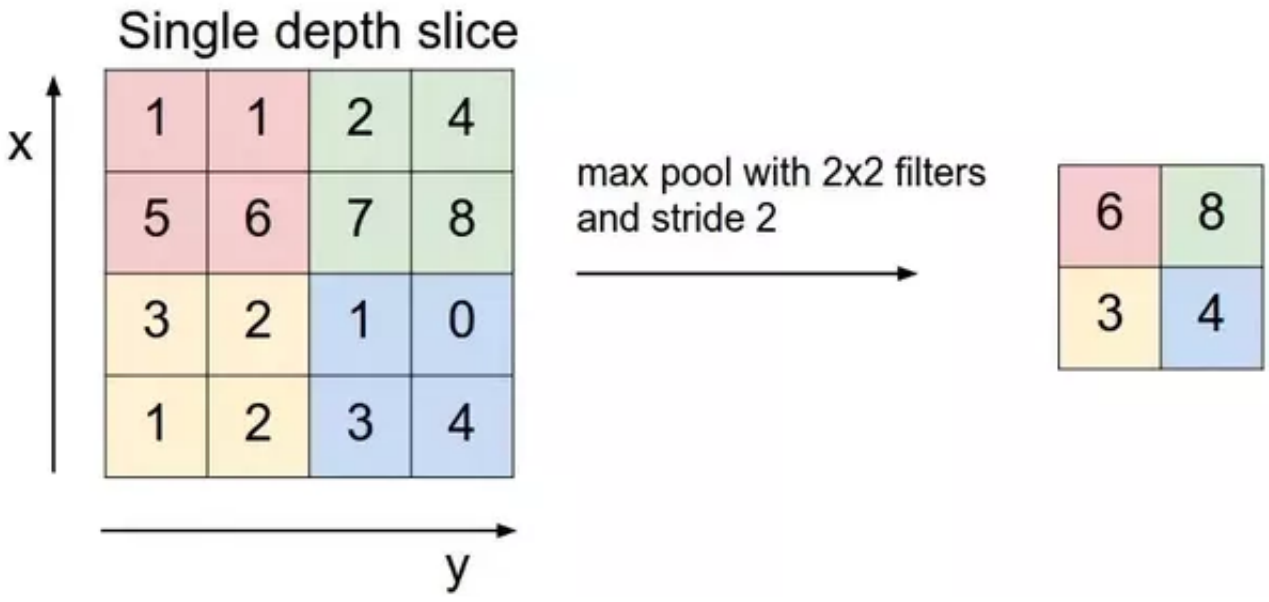

Figure 2.5: Max Pooling

2.1.5 What is Fully connected Layer?

The layer we call as FC layer, we flattened our matrix into vector and feed it into a fully connected layer like a neural network.

In the above diagram, the feature map matrix will be converted as vector (x1, x2, x3, …). With the fully connected layers, we combined these features together to create a model. Finally, we have an activation function such as SoftMax or sigmoid to classify the outputs as cat, dog, car, truck etc.

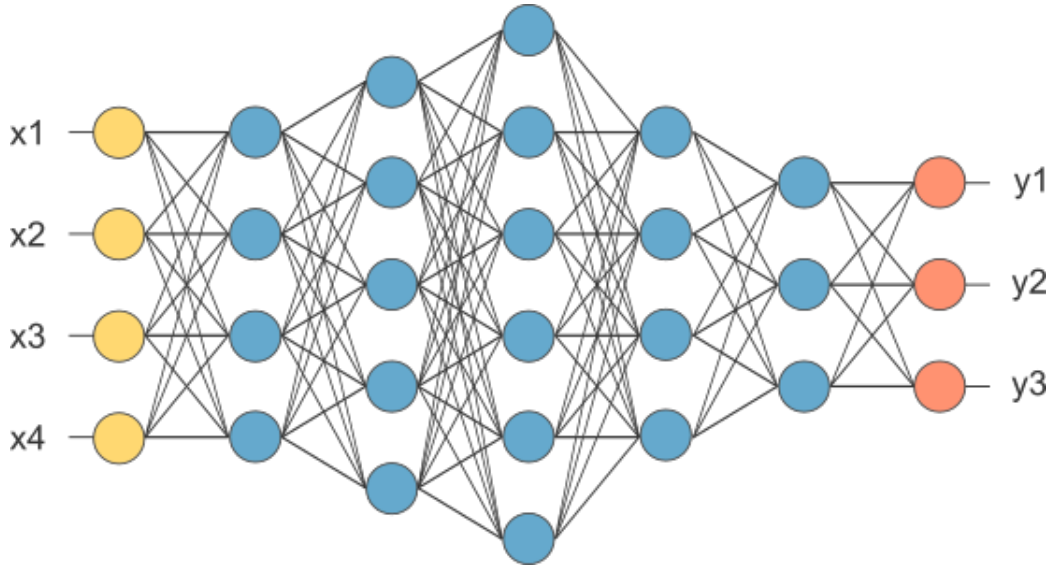

Figure 2.6: After pooling layer, flattened as FC layer

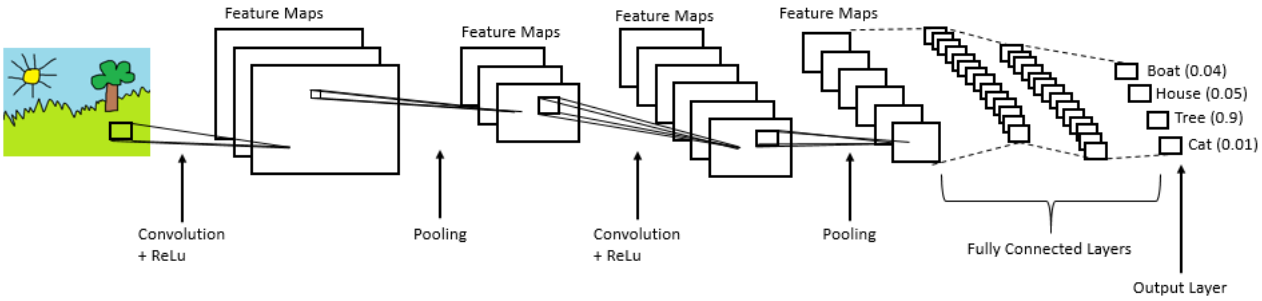

Figure 2.7: Complete CNN network

2.2 Activation Function

In a neural network, the activation function is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input. The rectified linear activation function is a piecewise linear function that will output the input directly if is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.

2.3 Sigmoid Function

The sigmoid function, also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

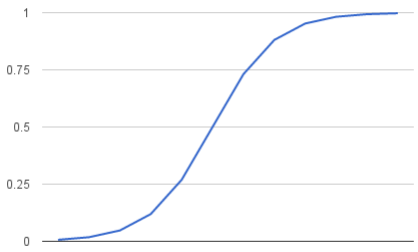

Figure 2.8: Sigmoid Function

2.3.1 Representation Used for Sigmoid Regression

Sigmoid regression uses an equation as the representation, very much like linear regression. Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modeled is a binary values (0 or 1) rather than a numeric value The actual representation of the model that you would store in memory or in a file are the coefficients in the equation (the beta value or b's).

2.3.2 Learning the Sigmoid Regression Model

The coefficients (Beta values b) of the sigmoid regression algorithm must be estimated from your training data. This is done using maximum-likelihood estimation. Maximum-likelihood estimation is a common learning algorithm used by a variety of machine learning algorithms, although it does make assumptions about the distribution of your data (more on this when we talk about preparing your data). The best coefficients would result in a model that would predict a value very close to 1 (e.g. male) for the default class

and a value very close to 0 (e.g. female) for the other class. The intuition for maximum-likelihood for sigmoid regression is that a search procedure seeks values for the coefficients (Beta values) that minimize the error in the probabilities predicted by the model to those in the data (e.g. probability of 1 if the data is the primary class).

2.3.3 Limitations of Sigmoid Activation Function

A neural network is comprised of layers of nodes and learns to map examples of inputs to outputs. For a given node, the inputs are multiplied by the weights in a node and summed together. This value is referred to as the summed activation of the node. The summed activation is then transformed via an activation function and defines the specific output or "activation" of the node. The simplest activation function is referred to as the linear activation, where no transform is applied at all. A network comprised of only linear activation functions is very easy to train, but cannot learn complex mapping functions. Linear activation functions are still used in the output layer for networks that predict a quantity (e.g. regression problems). Nonlinear activation functions are preferred as they allow the nodes to learn more complex structures in the data. Traditionally, two widely used nonlinear activation functions are the sigmoid and hyperbolic tangent activation functions. The sigmoid activation function, also called the sigmoid function, is traditionally a very popular activation function for neural networks. The input to the function is transformed into a value between 0.0 and 1.0. Inputs that are much larger than 1.0 are transformed to the value 1.0, similarly, values much smaller than 0.0 are snapped to 0.0. The shape of the function for all possible inputs is an S-shape from zero up through 0.5 to 1.0. For a long time, through the early 1990s, it was the default activation used on neural networks. The hyperbolic tangent function, or tanh for short, is a similar shaped nonlinear activation function that outputs values between -1.0 and 1.0. In the later 1990s and through the 2000s, the tanh function was preferred over the sigmoid activation function as models that used it were easier to train and often had better predictive performance.

2.4 Rectified Linear Activation Function

In order to use stochastic gradient descent with backpropagation of errors to train deep neural networks, an activation function is needed that looks and acts like a linear function, but is, in fact, a nonlinear function allowing complex relationships in the data to be learned. The function must also provide more sensitivity to the activation sum input and avoid easy saturation. The solution had been bouncing around in the field for some time, although was not highlighted until papers in 2009 and 2011 shone a light on it. The solution is to use the rectified linear activation function, or ReL for short. A node or unit that implements this activation function is referred to as a rectified linear activation unit, or ReLU for short. Often, networks that use the rectifier function for the hidden layers are referred to as rectified networks. Adoption of ReLU may easily be considered one of the few milestones in the deep learning revolution, e.g. the techniques that now permit the routine development of very deep neural networks.

2.4.1 Advantages of the Rectified Linear Activation Function

The rectified linear activation function has rapidly become the default activation function when developing most types of neural networks.

1. Computational Simplicity: The rectifier function is trivial to implement, requiring a max() function. This is unlike the tanh and sigmoid activation function that require the use of an exponential calculation.

2. Representational Sparsity: An important benefit of the rectifier function is that it is capable of outputting a true zero value. This is unlike the tanh and sigmoid activation functions that learn to approximate a zero output, e.g. a value very close to zero, but not a true zero value. This means that negative inputs can output true zero values allowing the activation of hidden layers in neural networks to contain one or more true zero values. This is called a sparse representation and is a desirable property in representational learning as it can accelerate learning and simplify the model. An area where efficient representations such as sparsity are studied and sought is in autoencoders, where a network learns a compact representation of an input (called the code layer), such as an image or series, before it is reconstructed from the compact representation.

3. Linear Behavior: The rectifier function mostly looks and acts like a linear acti-

vation function. In general, a neural network is easier to optimize when its behavior is linear or close to linear.

4.Train Deep Networks: Importantly, the (re-)discovery and adoption of the rectified linear activation function meant that it became possible to exploit improvements in hardware and successfully train deep multi-layered networks with a nonlinear activation function using backpropagation. In turn, cumbersome networks such as Boltzmann machines could be left behind as well as cumbersome training schemes such as layer-wise training and unlabeled pre-training.

Chapter 3

Proposed Work

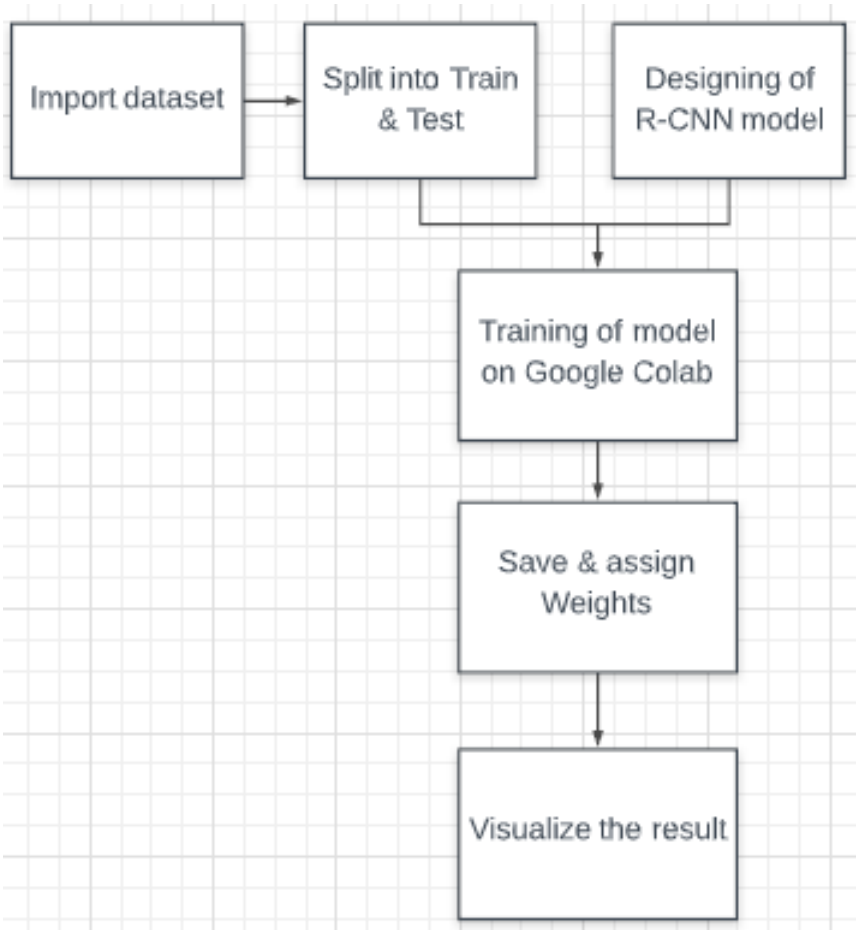

Figure 3.1: Flowchart

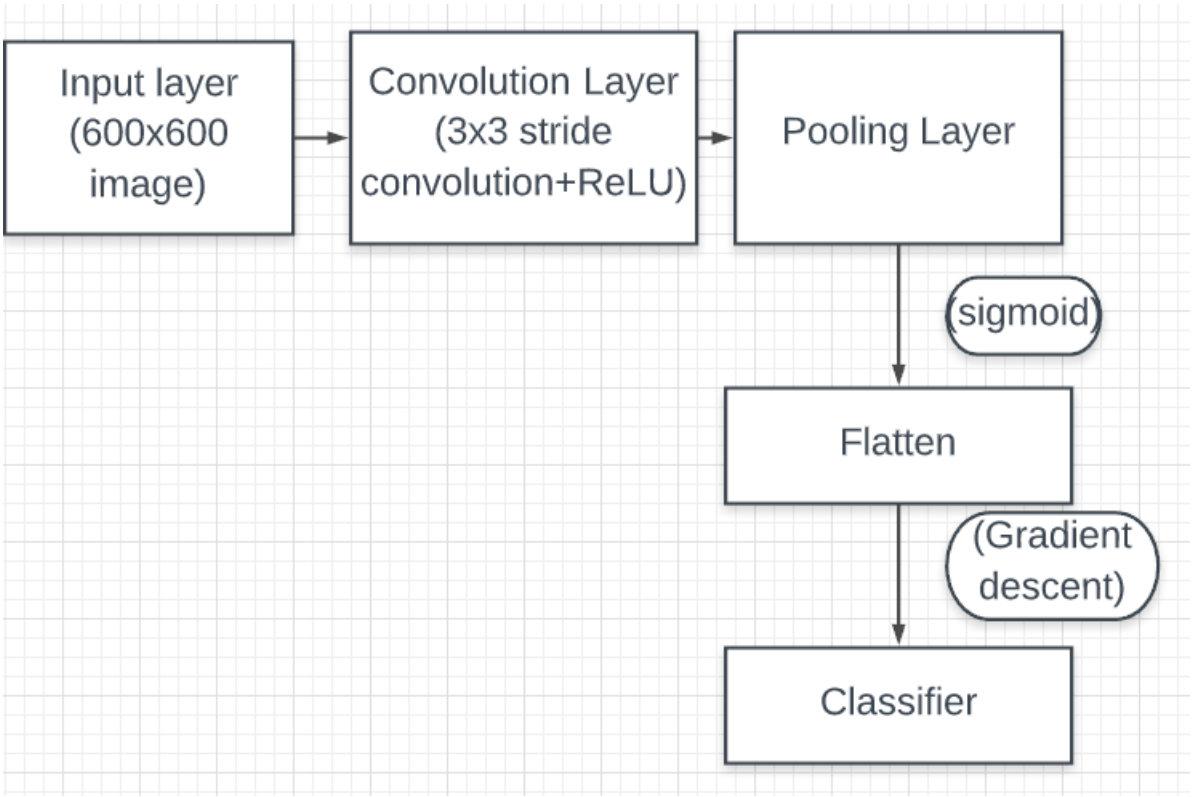

Figure 3.2: Model Structure

3.1 Binarization

3.1.1 Image Acquisition

The process of selecting the image and giving to system as an input is called image acquisition. We are using webcam for this purpose. The camera used is webcam of model no.C310 HD. Quality of acquiring image is one of key technology depends on camera. When the camera devices are determined, it is crucial to provide proper illumination approaches to ensure image quality. When extracting images, the luminary intensity and the stability of the auxiliary light source have great impacts on the image quality there are three kinds of camera data transmitting modes: USB, Camera Link, Gigabit Ethernet ports. We have used USB cable. Images captured by camera are RGB form, so it has to convert in grayscale as it is necessary for image pre-processing.

3.1.2 Image pre-processing

The two-dimensionalRGB image is converted intograyscale image. Each of R, G and B colour pixel has a rangeof [0,255]. If all the three colour pixels are merged together andhave a same pixelvalue at corresponding coordinates, itresults into grayscale image. It is calculating as,(R+G+B)/3. Grayscale imageis subjectedto geta binary image.Binary image is treated as black and white image in terms of0's and 1's. Set thethreshold value for interpretationof thepixel value. If the pixel value is greater than threshold then setthe pixel with the value 1(white) and replaces all other pixelwith 0(black)

3.1.3 XOR Operation

After the image conversion, apply bitwise XOR operation on two images which are in binary form to identify the defects. XOR operationis onlytrue ifone ofthe inputvalues is true otherwise false.

3.1.4 Drawback of Binarization

This method is widely used in industries. For this method we need picture of original PCB which is faultless. This image is given as INPUT to this system. But the problem arises when we don't have picture of original PCB. For this problem, the solution proposed by us is using MASK RCNN method.

3.2 Masked RCNN using Tensorflow GPU

3.2.1 How to Install Mask RCNN for Keras

Object detection is a task in computer vision that involves identifying the presence, location, and type of one or more objects in a given image.It is a challenging problem that involves building upon methods for object recognition (e.g. where are they), object localization (e.g. what are their extent), and object classification (e.g. what are they). The Region-Based Convolutional Neural Network, or RCNN, is a family of convolutional neural network models designed for object detection, developed by Ross Girshick, et al. There are perhaps four main variations of the approach, resulting in the current pinnacle called Mask RCNN. The Mask R-CNN introduced in the 2018 paper titled "Mask RCNN"

is the most recent variation of the family of models and supports both object detection and object segmentation. Object segmentation not only involves localizing objects in the image but also specifies a mask for the image, indicating exactly which pixels in the image belong to the object.Mask RCNN is a sophisticated model to implement, especially as compared to a simple or even state-of-the-art deep convolution neural network model. Instead of developing an implementation of the RCNN or Mask RCNN model from scratch, we can use a reliable third-party implementation built on top of the Keras deep learning framework.

Step 1: Clone the Mask R-CNN GitHub Repository. A repository is like a folder for your project. Your project's repository contains all of your project's files and stores each file's revision history. You can also discuss and manage your project's work within the repository. ... Public repositories are visible to everyone. t can be local to a folder on your computer, or it can be a storage space on GitHub or another online host. You can keep code files, text files, image files, you name it, inside a repository. This will create a new local directory with the name Mask RCNN

Step 2: Install the Mask R-CNN Library. The library can be installed directly via pip.

Step 3: Confirm the Library Was Installed. It is always a good idea to confirm that the library was installed correctly. You can confirm that the library was installed correctly by querying it via the pip command; for example: pip show mask-rcnn

3.2.2 How to Prepare a Dataset for Object Detection

The Mask RCNN is designed to learn to predict both bounding boxes for objects as well as masks for those detected objects, and the MISSING HOLE dataset does not provide masks. As such, we will use the dataset to learn a MISSING HOLE object detection task, and ignore the masks and not focus on the image segmentation capabilities of the model. There are a few steps required in order to prepare this dataset for modelling and we will work through each in turn in this section, including downloading the dataset, parsing the annotations file, developing a MISSING HOLE Dataset object that can be used by the Mask RCNN library, then testing the dataset object to confirm that we are loading images and annotations correctly.

Step 1: Install Dataset The first step is to download the dataset into your current

working directory. This can be achieved by cloning the GitHub repository directly, as follows: git clone https: github.com experiencor missing hole.git. This will create a new directory called “missing hole” with a subdirectory called ‘images’ that contains all of the JPEG photos of missing holes and a subdirectory called ‘annotes/‘ that contains all of the XML files that describe the locations of missing holes in each photoThis means that we should focus on loading the list of actual files in the directory rather than using a numbering system.

Step 2: Parse Annotation File We can see that the annotation file contains a “size” element that describes the shape of the photograph, and one or more “object” elements that describe the bounding boxes for the missing hole objects in the photograph. The size and the bounding boxes are the minimum information that we require from each annotation file. We could write some careful XML parsing code to process these annotation files, and that would be a good idea for a production system. Instead, we will short-cut development and use XPath queries to directly extract the data that we need from each file, e.g. a size query to extract the size element and a object or a bndbox query to extract the bounding box elements. Python provides the ElementTree API that can be used to load and parse an XML file and we can use the find() and findall() functions to perform the XPath queries on a loaded document.

Step 3: Develop MISSING HOLE Dataset Object The mask-rcnn library requires that train, validation, and test datasets be managed by a mrcnn.utils.Dataset object. This means that a new class must be defined that extends the mrcnn.utils.Dataset class and defines a function to load the dataset, with any name you like such as load dataset(), and override two functions, one for loading a mask called load mask() and one for loading an image reference (path or URL) called image reference(). To use a Dataset object, it is instantiated, then your custom load function must be called, then finally the built-in prepare() function is called. The custom load function, e.g. load dataset() is responsible for both defining the classes and for defining the images in the dataset. Classes are defined by calling the built-in add class() function and specifying the ‘source‘ (the name of the dataset), the ‘class id‘ or integer for the class (e.g. 1 for the first lass as 0 is reserved for the background class), and the ‘class name‘ such as missing hole. This will define an “image info” dictionary for the image that can be retrieved later via the index or order in which the image was added to the dataset. You can also specify other arguments that will

be added to the image info dictionary, such as an ‘annotation‘ to define the annotation path.

Step 4: Test MISSING HOLE Dataset Object The first useful test is to confirm that the images and masks can be loaded correctly. We can test this by creating a dataset and loading an image via a call to the load image() function with an image id, then load the mask for the image via a call to the load mask() function with the same image id. Next, we can plot the photograph using the Matplotlib API, then plot the first mask over the top with an alpha value so that the photograph underneath can still be seen. Running the example first prints the shape of the photograph and mask NumPy arrays. We can confirm that both arrays have the same width and height and only differ in terms of the number of channels. We can also see that the first photograph (e.g. image id=0). In this case only has one mask Finally, the mask-rcnn library provides utilities for displaying images and masks. We can use some of these built-in functions to confirm that the Dataset is operating correctly. For example, the mask-rcnn library provides the mrcnn.visualize.display instances() function that will show a photograph with bounding boxes, masks, and class labels. This requires that the bounding boxes are extracted from the masks via the extract bboxes() function.

3.2.3 How to Train Mask RCNN Model for Missing Hole Detection

A Mask RCNN model can be fit from scratch, although like other computer vision applications, time can be saved and performance can be improved by using transfer learning. The Mask RCNN model pre-fit on the MS COCO object detection dataset can be used as a starting point and then tailored to the specific dataset, in this case, the MISSING HOLE dataset. The first step is to download the model file (architecture and weights) for the pre-fit Mask RCNN model. The weights are available from the GitHub project and the file is about 250 megabytes. Next, a configuration object for the model must be defined. This is a new class that extends the mrcnn.config. Config class and defines properties of both the prediction problem (such as name and the number of classes) and the algorithm for training the model (such as the learning rate). The configuration must define the name of the configuration via the ‘NAME‘ attribute, e.g. ‘missing hole cfg‘, that will be used to save details and models to file during the run. The configuration must also define the number of classes in the prediction problem via the ‘NUM CLASSES‘ attribute. In

this case, we only have one object type of hole, although there is always an additional class for the background. Finally, we must define the number of samples (photos) used in each training epoch. This will be the number of photos in the training dataset.

3.2.4 How to Evaluate a Mask RCNN Model

The performance of a model for an object recognition task is often evaluated using the mean absolute precision, or MAP. We are predicting bounding boxes so we can determine whether a bounding box prediction is good or not based on how well the predicted and actual bounding boxes overlap. This can be calculated by dividing the area of the overlap by the total area of both bounding boxes, or the intersection divided by the union, referred to as "intersection over union," or IoU. A perfect bounding box prediction will have an IoU of 1. It is standard to assume a positive prediction of a bounding box if the IoU is greater than 0.5, e.g. they overlap by 50 percent or more. Precision refers to the percentage of the correctly predicted bounding boxes (IoU > 0.5) out of all bounding boxes predicted. Recall is the percentage of the correctly predicted bounding boxes (IoU > 0.5) out of all objects in the photo. As we make more predictions, the recall percentage will increase, but precision will drop or become erratic as we start making false positive predictions. The recall (x) can be plotted against the precision (y) for each number of predictions to create a curve or line. We can maximize the value of each point on this line and calculate the average value of the precision or AP for each value of recall.

The average or mean of the average precision (AP) across all of the images in a dataset is called the mean average precision, or MAP.

The Mask RCNN library provides a mrcnn.utils.compute ap to calculate the AP and other metrics for a given images. These AP scores can be collected across a dataset and the mean calculated to give an idea at how good the model is at detecting objects in a dataset. First, we must define a new Config object to use for making predictions, instead of training. We can extend our previously defined MissingholeConfig to reuse the parameters. Instead, we will define a new object with the same values to keep the code compact. The config must change some of the defaults around using the GPU for inference that are different from how they are set for training a model (regardless of whether you are running on the GPU or CPU).

3.2.5 How to detect missing holes in new pictures

We can use the trained model to detect missing holes in new photographs, specifically, in photos that we expect to have missing holes. First, we need a new photo of a PCB. We could go to Flickr and find a random photo of a PCB. Alternately, we can use any of the photos in the test dataset that were not used to train the model. We have already seen in the previous section how to make a prediction with an image. Specifically, scaling the pixel values and calling model.detect(). Let's take it one step further and make predictions for a number of images in a dataset, then plot the photo with bounding boxes side-by-side with the photo and the predicted bounding boxes. This will provide a visual guide to how good the model is at making predictions. The first step is to load the image and mask from the dataset Next, we can make a prediction for the image. Next, we can create a subplot for the ground truth and plot the image with the known bounding boxes. We can then create a second subplot beside the first and plot the first, plot the photo again, and this time draw the predicted bounding boxes in red. We can tie all of this together into a function that takes a dataset, model, and config and creates a plot of the first five photos in the dataset with ground truth and predicted bound boxes.

Chapter 4

Results

4.1 Parameter Tables

Table 4.1: Layer Structure

Layer Name	Standard weight
conv1.2/kernel	0.1315
conv1.2/bias	0. 0.0001
res2a.branch2a.2/kernel	0.5091
res2a.branch2a.2/bias	1.9781
res2a.branch2b.2/kernel	44.5654
res2a.branch2b.2/bias	2184.7
res2a.branch2c.2/kernel	0.0764
res2a.branch2c.2/bias	0.0018
res2a.branch1.2/kernel	0.4116

Table 4.2: Comparison of Accuracy

Model Type	Train Accuracy	Test Accuracy
CNN	88.7	76.3
RCNN	91.6	78.5
MASKED RCNN	98.8	98.7

Table 4.3: Loss per Epoch Cycle

Number of Epochs	Model Loss	rpn class loss	rpn bbox loss	mrcnn class loss	val loss
0 to 5	0.9819	0.0146	0.2742	0.069	0.7498
5 to 10	0.6909	0.0007	0.252	0.0263	0.7263
10 to 15	0.4552	0.0007	0.1717	0.0114	0.5017

4.2 Output Images

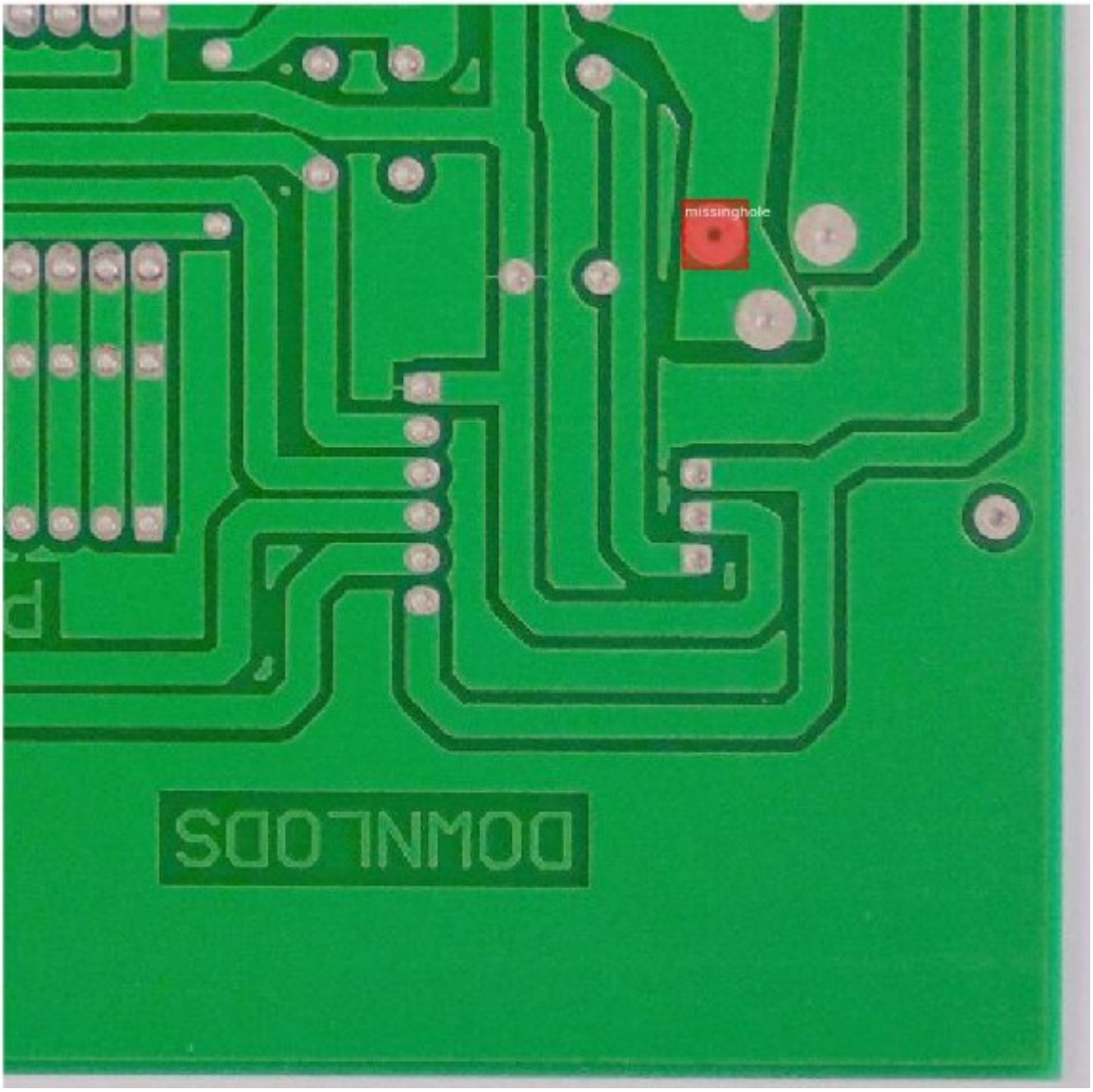

Figure 4.1: Masking of rcnn

```
In [12]: runfile('E:/B.Tech/Prjkt_Mjr/datasetprep.py', wdir='E:/B.Tech/Prjkt_Mjr')
Train: 1282
Test: 550

In [13]:
```

Figure 4.2: Data preparation

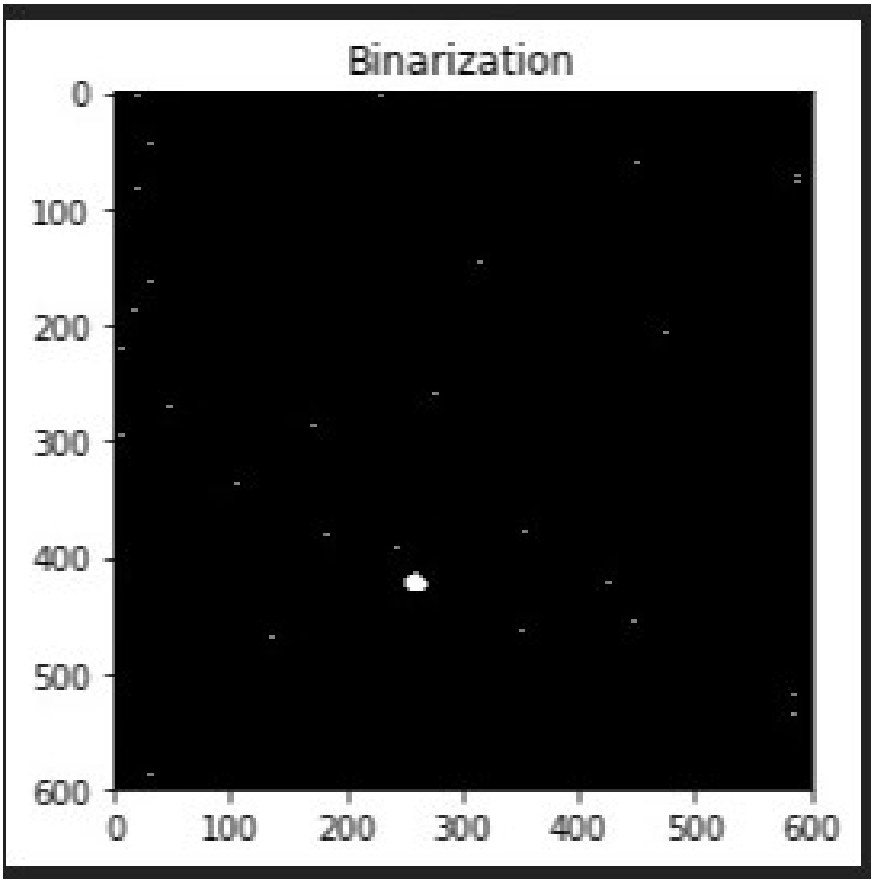

Figure 4.3: Output of binarization

```
[26] # create config
     cfg = PredictionConfig()
     # define the model
     model = MaskRCNN(mode='inference', model_dir='./new.h5', config=cfg)
     # load model weights
     model.load_weights('new.h5', by_name=True)

[27] # evaluate model on training dataset
     train_mAP = evaluate_model(train_set, model, cfg)
     print("Train mAP: %.3f" % train_mAP)
     # evaluate model on test dataset
     test_mAP = evaluate_model(test_set, model, cfg)
     print("Test mAP: %.3f" % test_mAP)

     Train mAP: 0.988
     Test mAP: 0.987
```

Figure 4.4: Result MAP's

```
Epoch 1/5
1282/1282 [==============================] - 1524s 1s/step - loss: 0.9819 - rpn_class_loss: 0.0146 - rpn_bbox_loss: 0.2742 - mrcnn_class_loss: 0.0696 -
Epoch 2/5
1282/1282 [==============================] - 1451s 1s/step - loss: 0.7769 - rpn_class_loss: 0.0109 - rpn_bbox_loss: 0.2679 - mrcnn_class_loss: 0.0342 -
Epoch 3/5
1282/1282 [==============================] - 1457s 1s/step - loss: 0.6909 - rpn_class_loss: 0.0070 - rpn_bbox_loss: 0.2520 - mrcnn_class_loss: 0.0263 -
Epoch 4/5
1282/1282 [==============================] - 1448s 1s/step - loss: 0.6753 - rpn_class_loss: 0.0071 - rpn_bbox_loss: 0.2482 - mrcnn_class_loss: 0.0240 -
Epoch 5/5
1282/1282 [==============================] - 1442s 1s/step - loss: 0.6165 - rpn_class_loss: 0.0066 - rpn_bbox_loss: 0.2247 - mrcnn_class_loss: 0.0172 -
```

Figure 4.5: Epoch Cycles

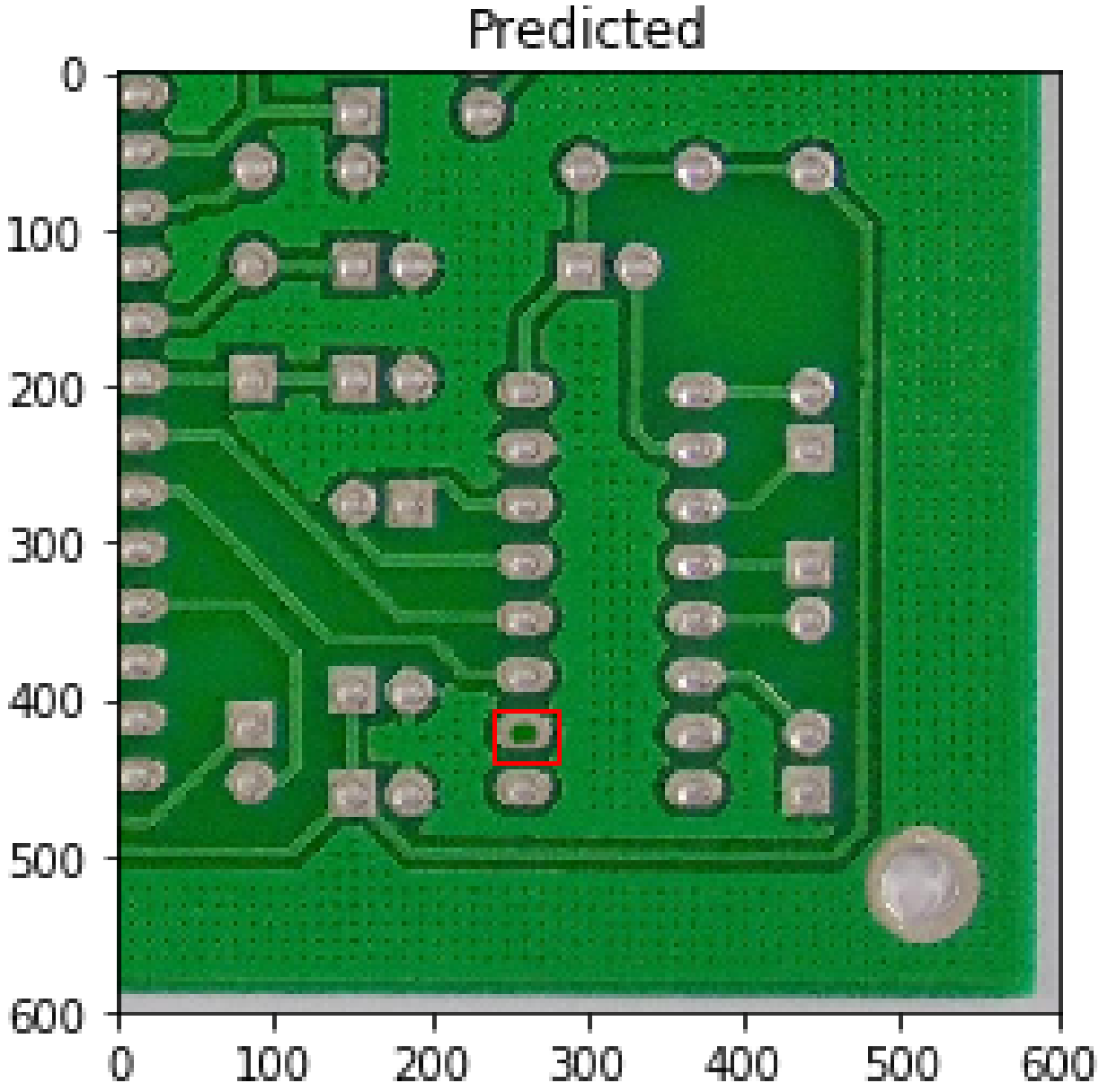

Figure 4.6: Local prediction 1

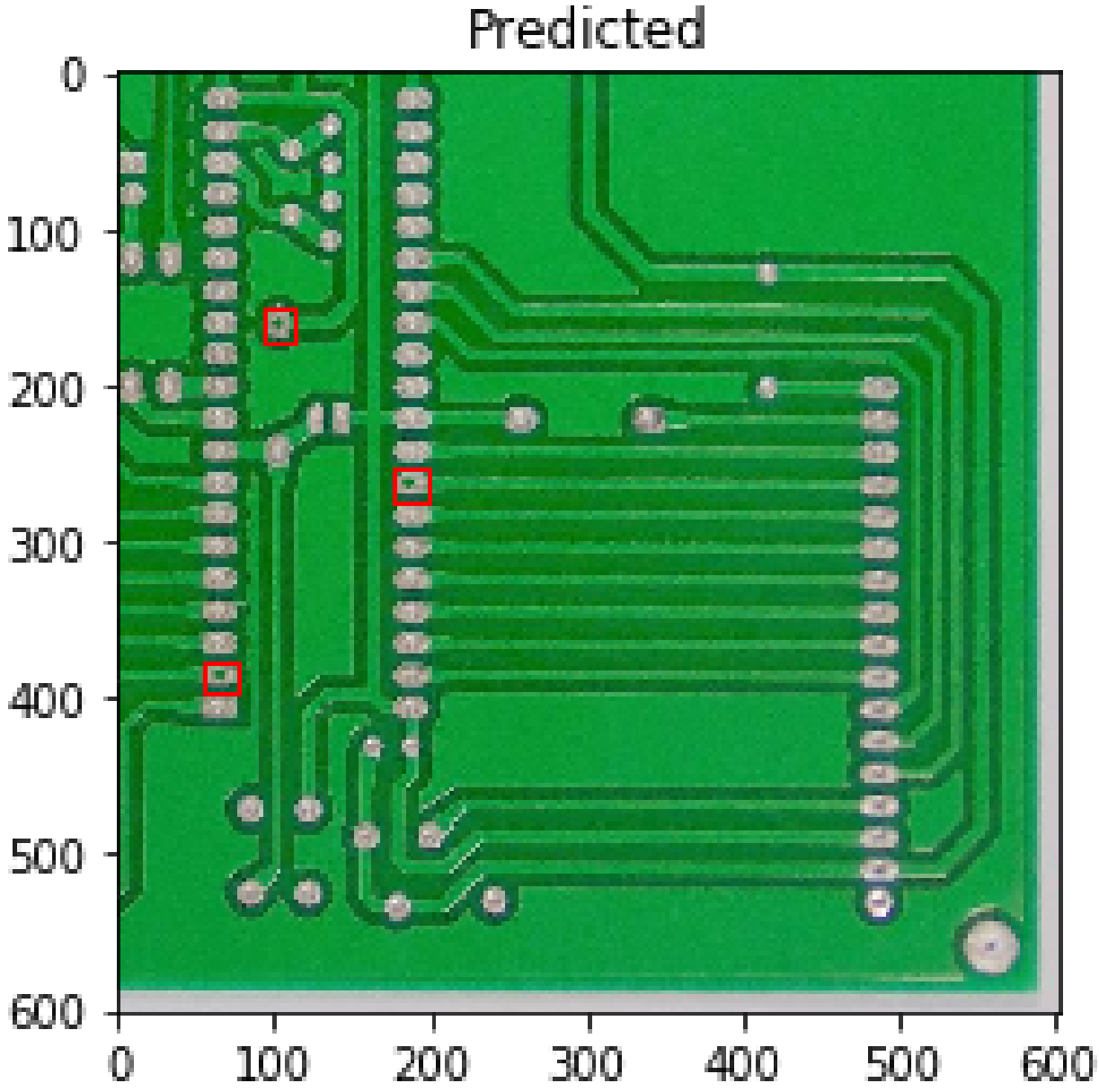

Figure 4.7: Local prediction 2

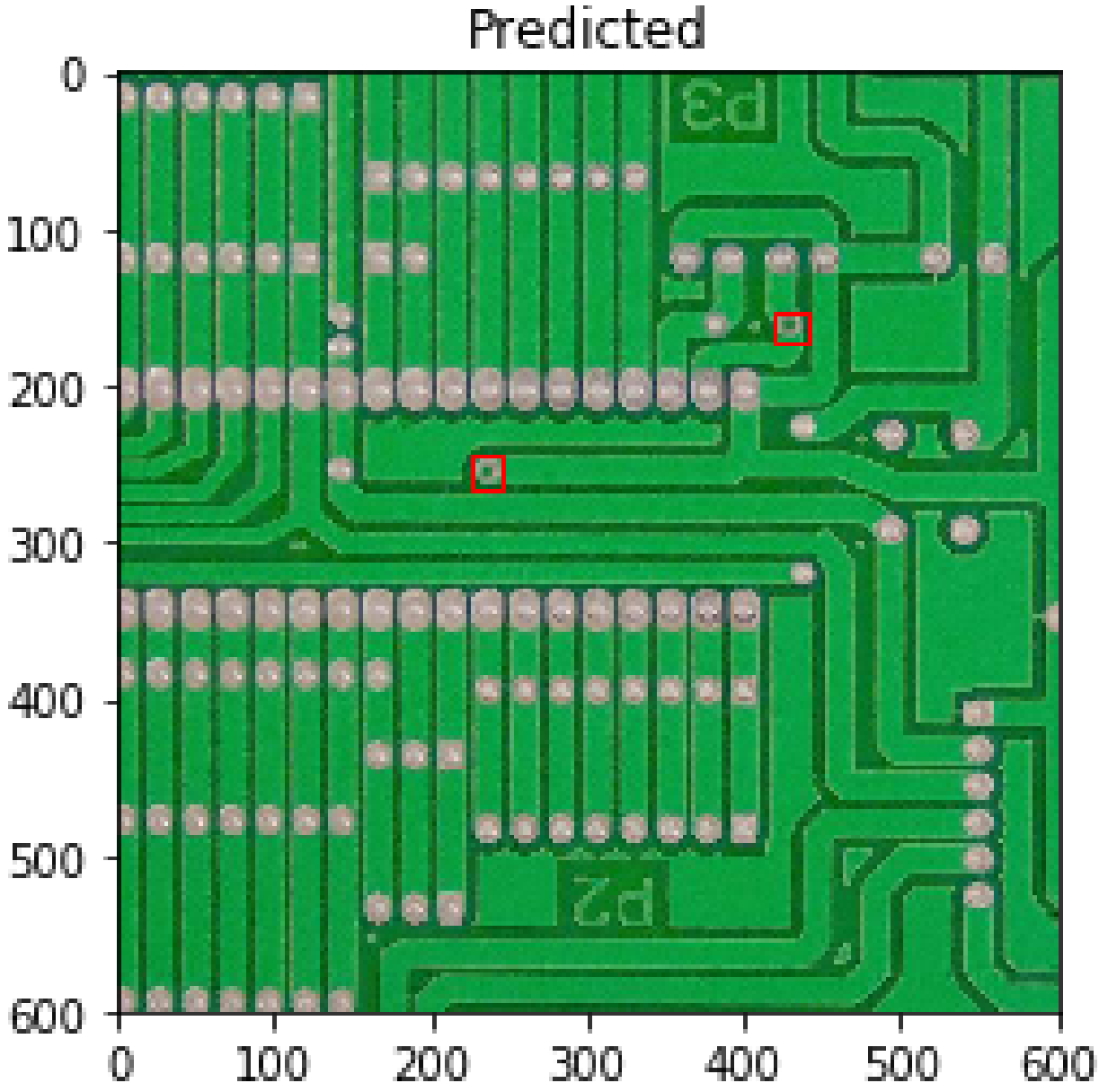

Figure 4.8: Local prediction 3

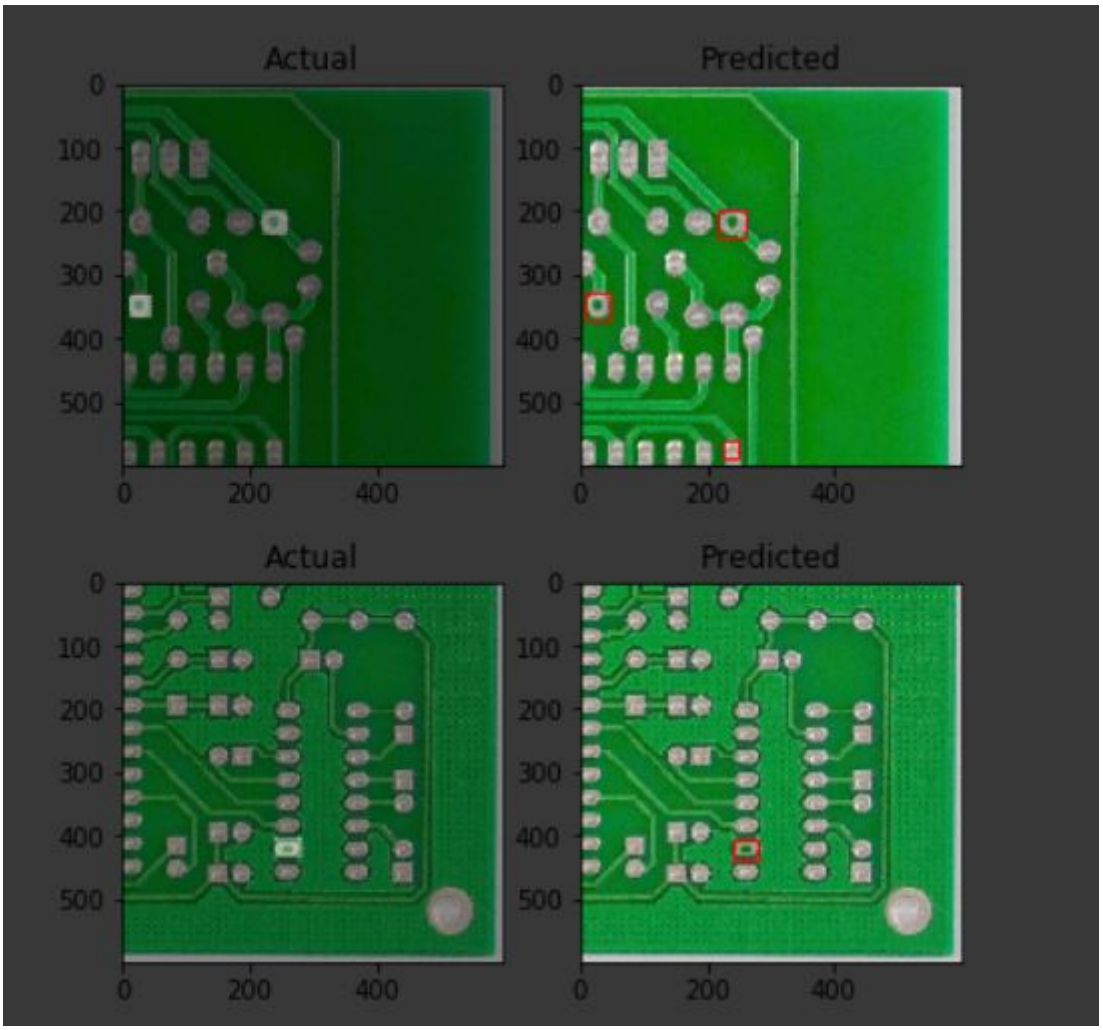

Figure 4.9: Train and Test Prediction Comparision

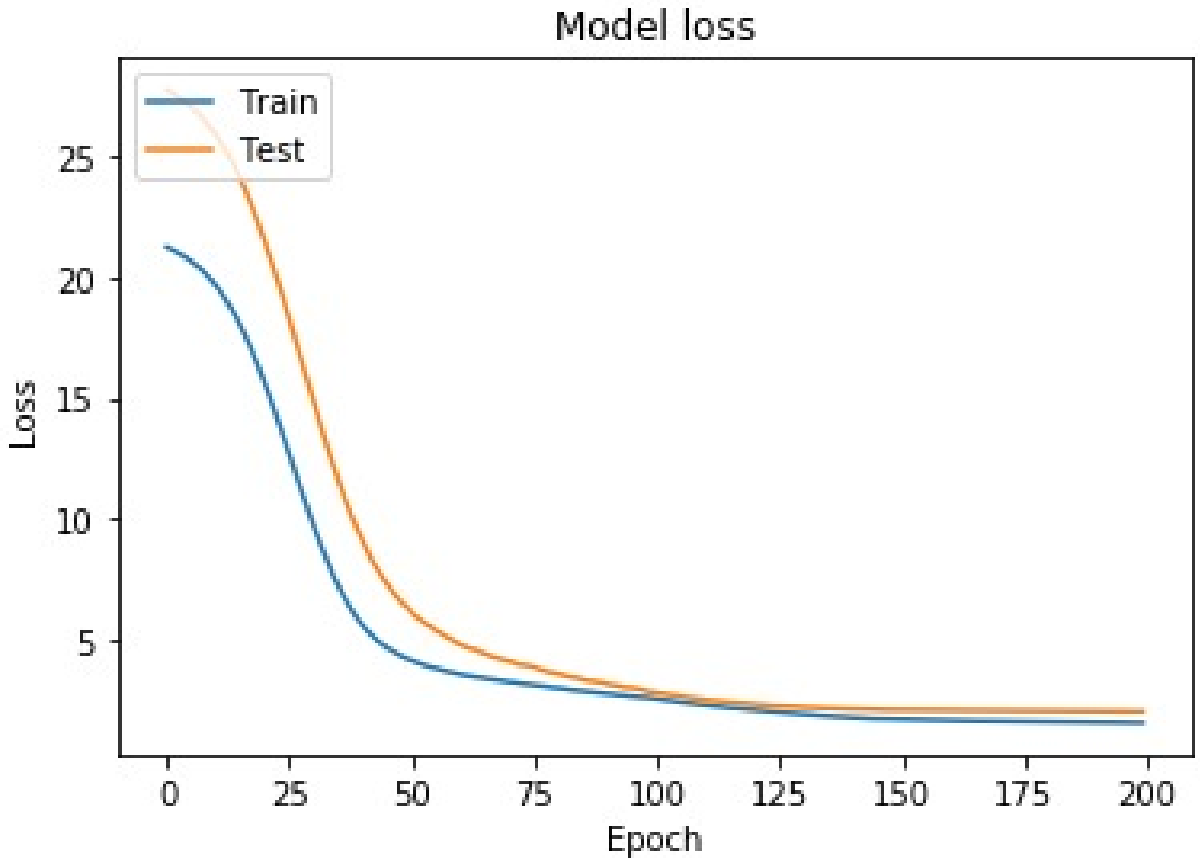

Figure 4.10: Model Loss Per Epoch

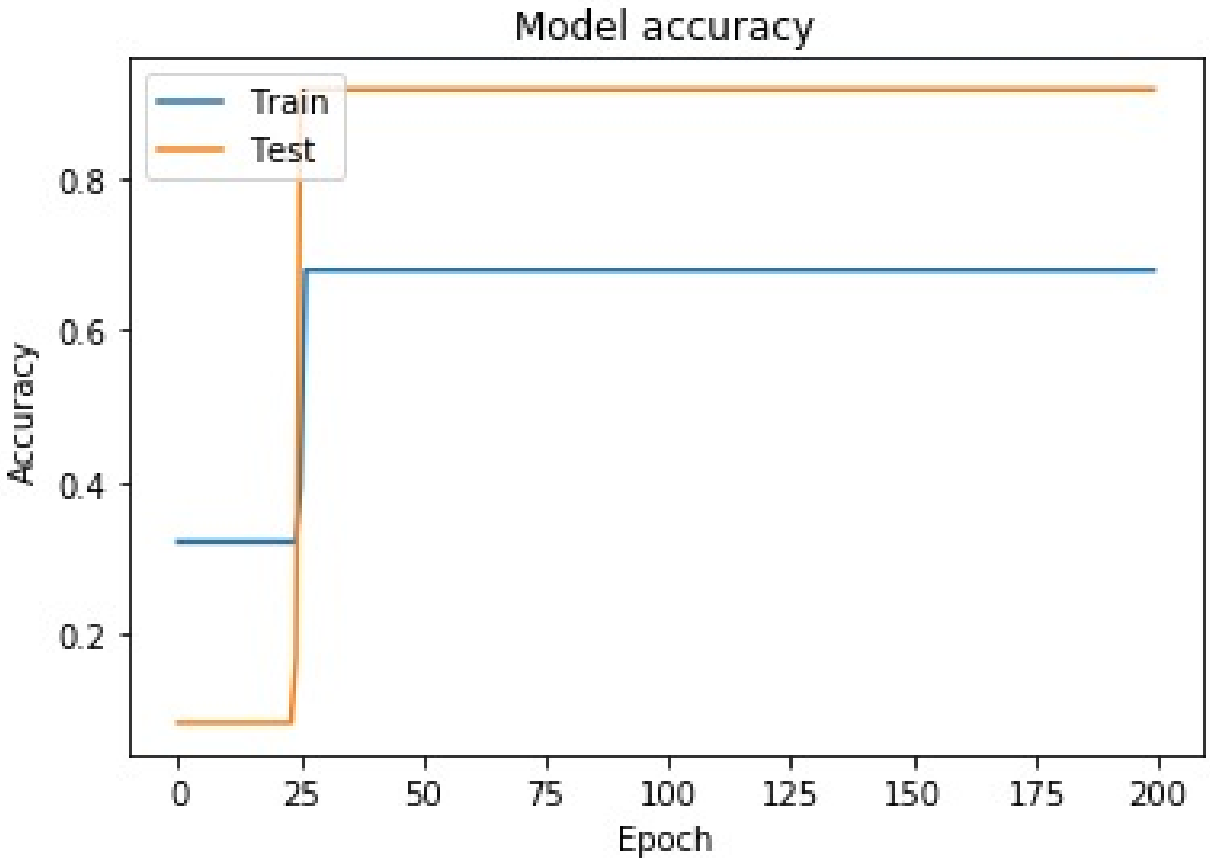

Figure 4.11: Model Accuracy per Epoch

Chapter 5

Conclusion

Object detection is a challenging computer vision task that involves predicting both where the objects are in the image and what type of objects were detected. R-CNN algorithms have truly been a game-changer for object detection tasks. There has suddenly been a spike in recent years in the amount of computer vision applications being created, and R-CNN is at the heart of most of them. In the above Project we have proposed a methodology for Fault detection in a printed circuit board. The Model was designed and trained for 243 layers of a masked R-CNN and was trained using a GPU on google-colab. The weights were saved in a HyPy file and were used to load the model afterwards. The performance of the model was evaluated using both the test set as reference as well as the real-life image of a printed circuit board that was captured using webcam. The local model was very successful on the test set with accuracy of 98.7. In conclusion, the model is working perfectly fine for detection of missing holes on a printed circuit board. This, model can be used in circuit industry for detecting the faulty printed circuit boards. If a GPU is used this could work way faster than the traditional methods and would be more accurate.

5.1 Future Scope

The wide use of vision systems is getting very common in all aspects of life. The job of testing a product involves various aspects and considerations. This job is tedious and slow to do. But, with fast working vision systems it becomes really easy, robust and fast paced. The project has a very realistic future scope as the model was trained only

for missing hole detection it could well be trained for various different errors and faults on printed circuit boards. Also, if graphical processing unit is available then it could be used with point efficiency and large scale with faster production lines. If the graphical processing unit is available for use the training process could be speeded up. The current model is only a feedforward learning network but it could well be compiled for a feedback propagation style learning this could enable the learn on-the go and could be very useful for the increasing the accuracy.

Bibliography

[1] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. Eccv, 110(3):404–417, 2006.

[2] Vikas Chaudhary, Ishan R. Dave, and Kishor P. Upla. Automatic visual inspection of printed circuit board for defect detection and classification. Proceedings of the 2017 International Conference on Wireless Communications, Signal Processing and Networking, WiSPNET 2017, 2018-Janua:732–737, 2018.

[3] Shahrzad Faghih-Roohi, Hajizadeh Siamak, Alfredo Nez, Robert Babuska, and Bart De Schutter. Deep convolutional neural networks for detection of rail surface defects. In International Joint Conference on Neural Networks, 2016.

[4] Xavier Glorot, Antoine Bordes, and Y Bengio. Deep sparse rectifier neural networks. Proceedings of the 14th International Conference on Artificial Intelligence and Statisitics (AISTATS) 2011, 15:315–323, 01 2011.

[5] Rafael C. Gonzalez, Richard E. Woods, and Steven L. Eddins. Digital Image Processing Using Matlab. Publishing House of Electronics Industry, 2009.

[6] M. Guo and R. Wang, "Research of the Machine Vision Based PCB Defect Inspection System," International Conference on Intelligence Science and Information Engineering, Wuhan, 2011, pp. 472-475.

[7] L. z. Lin, L. s. Zhou, J. d. Wan et al., "Study of PCB Automatic Optical Inspection System Based on Mathematical Morphology," International Conference on Computer Technology and Development, Kota Kinabalu, 2009, pp. 405-408.

[8] C. Benedek, "Detection of soldering defects in printed circuit boards with hierarchical marked point processes," Pattern Recognition Letters, 2011, 32, (13) pp. 1535- 1543.

[9] K. Fukushima and S. N. Miyake, “A new algorithm for pattern recognition tolerant of deformations and shifts in position,” Pattern recognition, 1982, 15, (6) pp. 455- 469.

[10] J. Ma, “Defect detection and recognition of bare PCB based on computer vision,” Chinese Control Conference (CCC), Dalian, 2017, pp. 11023-11028.

[11] Tae-Hyoung Park and Hwa-Jung Kim, ”Path planning of automatic optical inspection machines for PCB assembly systems,” 2005 International Symposium on Computational Intelligence in Robotics and Automation, 2005, pp. 249-254.

[12] Elias N. Malamas, Euripides G.M. Petrakis, Michalis. Zervakis, et al, “A survey on industrial vision systems, applications and tools,” Image and vision computing, vol. 21, no. 2, pp. 171–188, 2003.

[13] Madhav Moganti, Fikret Ercal, Cihan H. Dagli, et al, “Automatic PCB inspection algorithms: a survey,” Computer vision and image understanding, vol. 63, no. 2, pp. 287-313, 1996.

[14] Z. Ibrahim, S. A. R. Al-Attas, Z. Aspar, et al, ”Performance evaluation of wavelet-based PCB defect detection and localization algorithm,” Industrial Technology, 2002. IEEE ICIT ’02. 2002 IEEE International Conference on, 2002, pp. 226-231 vol.1.

[15] H. Rau and C. H. Wu, “Automatic optical inspection for detecting defects on printed circuit board inner layers,” International Journal of Advanced Manufacturing Technology, vol. 25, no. 9–10, pp. 940–946, 2005.