

**EXPLORING ARTIFICIAL INTELLIGENCE(AI)  
FOR MACHINE AUTOMATION**

**MERVYN MEOW ZI YANG**

**A project report submitted in partial fulfilment of the  
requirements for the award of the degree of  
Bachelor of Engineering (Hons) Electronic Engineering**

**Faculty of Engineering and Green Technology  
Universiti Tunku Abdul Rahman**

**May 2019**

## DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : \_\_\_\_\_

Name : \_\_\_\_\_

ID No. : \_\_\_\_\_

Date : \_\_\_\_\_

**APPROVAL FOR SUBMISSION**

I certify that this project report entitled **“EXPLORING ARTIFICIAL INTELLIGENCE (AI) FOR MACHINE AUTOMATION”** was prepared by **MERVYN MEOW ZI YANG** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons) Electronic Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : \_\_\_\_\_

Supervisor: Dr. Teh Peh Chiong

Date : \_\_\_\_\_

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of Universiti Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2018, Mervyn Meow Zi Yang. All right reserved.

## ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Dr. Teh Peh Chiong. Thanks for his invaluable advice, guidance and his enormous patience throughout the development of the research. His timely suggestions with enthusiasm, kindness, enthusiasm and dynamism have enable me to complete my thesis. Thanks for his keen interest which lead me on every stage of my research.

In addition, I would also like to express my gratitude to my loving parent and friends who had helped and given me encouragement. Their understanding, co-operation and constant encouragement are the sustaining factors in carrying out my work successfully.

I also express my deep sense of gratitude to my friend, Ms. Lim Siew Kee for her valuable suggestion and kindness in helping me throughout my research project.

## **EXPLORING ARTIFICIAL INTELLIGENCE(AI) FOR MACHINE AUTOMATION**

### **ABSTRACT**

Artificial intelligence (AI) is something intelligent and it could perform things that only human can perform. It might even be more powerful than the human minds if it was well developed. People all around the world are getting more familiar with AI as the technology development are getting more advance. Object detection task is one of the most popular example of artificial intelligence system that used to identify and classify objects. Inside the object detection task, it consists of deep convolutional neural networks as a classifier. This classifier is work together with other object detection technique to detect the region of interest of a particular image. There are many different type of open source frameworks such as Tensorflow, pytorch, Caffe and Keras are available on the internet. Many research had been done using Tensorflow by those huge company such as Nvidia, Uber and Snapchat in defecting object or face. Tensorflow is consider as low-level language which is more flexible in design. It is important to have more flexibility in desiging own functionalities as it allows us to change the architecture of networks based on our requirements. Researcher can understand how the operations are implemented through the network control provided by Tensorflow. It also allows the researcher to keep track of the updated change over certain time period. In this project, we use the Tensorflow Object Detection API which is an open source framework for object detection related task to identify and classify different types of components. Different type of deep learning models is used to make comparison in term of accuracy. In this case, we used Faster R-CNN as our object detection model and Inception-V2 as our feature extraction network. Faster R-CNN to run through the Region Proposal Network in order to obtain the region of interest and then input into the classifier network to obtain the classes for the particular object.

## TABLE OF CONTENTS

<b>DECLARATION</b>	<b>ii</b>
<b>APPROVAL FOR SUBMISSION</b>	<b>iii</b>
<b>ACKNOWLEDGEMENTS</b>	<b>v</b>
<b>ABSTRACT</b>	<b>vi</b>
<b>TABLE OF CONTENTS</b>	<b>vii</b>
<b>LIST OF TABLES</b>	<b>x</b>
<b>LIST OF FIGURES</b>	<b>xi</b>
<b>LIST OF APPENDICES</b>	<b>xiv</b>

### CHAPTER

<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
	1.1 Background	1
	1.2 Problem Statements	3
	1.3 Applications	5
	1.4 Aims and Objectives	6
	1.5 Scope	6
<b>2</b>	<b>LITERATURE REVIEW</b>	<b>8</b>
	2.1 Introduction	8
	2.1.1 Automated Optical Inspection (AOI) Machine	8
	2.2 Artificial Intelligence, Machine Learning and Deep Learning	9
	2.2.1 Machine Learning	9
	2.2.2 Deep Learning	11

2.3	Convolutional Neural Network	12
2.3.1	LeNet (Created by LeCun)	12
2.3.2	AlexNet	12
2.3.3	ZFNet (Created by Zeiler and Furgus)	13
2.3.4	GoogLeNet/Inception	13
2.3.5	VGGNet (Created by Visual Geometry Group)	14
2.3.6	ResNet (Residual Neural Network)	14
2.4	Basic Building Block of Convolutional Neural Network	15
2.4.1	Convolutional Layer	15
2.4.2	Adaptive activation funtion	16
2.4.3	Max Pooling Layer	18
2.4.4	Dropout	19
2.4.5	Optimization Algorithms	19
2.5	Object Detection Methods	21
2.5.1	Bounding Box	21
2.5.2	Anchor Boxes	23
2.5.3	Classification and Regression	24
2.6	Two-Stage Method	24
2.6.1	RCNN (Regional-Based Convolutional Neural Network)	24
2.6.2	Fast RCNN (Fast Regional-Based Convolutional Neural Network)	25
2.6.3	Faster RCNN (Faster Regional-Based Convolutional Neural Network)	26
2.7	Unified Method	27
2.7.1	YOLO (You Only Look Once)	27
2.7.2	SSD (Single Shot Detector)	27
2.8	Comparison of SSD with Faster RCNN	28
2.9	Preferred Coding Language	28
<b>3</b>	<b>METHODOLOGY</b>	<b>29</b>
3.1	Approach	29
3.2	Data Gathering and Labelling	29



3.3	Hardware and Software	30
3.4	Model Training Details	30
3.4.1	Faster RCNN	30
3.4.2	Inception Network	32
3.4.3	Network Setting	35
3.5	Experiment Setup	37
3.6	Testing	39
<b>4</b>	<b>RESULTS AND DISCUSSIONS</b>	<b>40</b>
4.1	Losses	40
4.1.1	Box Classifier Loss	40
4.1.2	RPN Loss	41
4.1.3	Total Loss and Clone Loss	42
4.2	Validation Data	43
4.2.1	Correct Detection	43
4.2.2	Error Detection	46
4.2.3	Unrecognized Data	49
4.3	Analysis	50
4.3.1	Mean Average Precision	50
4.3.2	Comparison of GPU Speed	53
4.3.3	Comparison of mean average precision using different amount of images	55
<b>5</b>	<b>CONCLUSION</b>	<b>57</b>
5.1	Conclusion	57
5.2	Future Implementation and Recommendation	58
	<b>REFERENCES</b>	<b>60</b>
	<b>APPENDICES</b>	<b>64</b>

**LIST OF TABLES**

<b>TABLE</b>	<b>TITLE</b>	<b>PAGE</b>
4.1	Average Precision (in percentage) for each Different Classes	51
4.2	Performance score for different type of GPUs	52
4.3	Comparison of time used per epoch for training using different GPUs	53
4.4	Mean Average Precision Obtained using Different Amount of Images for Training	54

## LIST OF FIGURES

FIGURE	TITLE	PAGE
2.1	Relationship between Artificial Intelligence, Machine Learning and Deep Learning	9
2.2.1	Input Image in Array Form	16
2.2.2	Learnable Filter with Certain Weight	16
2.2.3	Convolved Feature Map	16
2.3	ReLU activation function for $f(x) = \max(0, x)$	18
2.4	Max Pooling Process	19
2.5	Gradient Descent Model	20
2.6.1	Computing the Intersection of Union	22
2.6.2	Example of Computed Intersection of Union	22
2.6.3	Anchor Boxes	23
2.7.1	RCNN Model	25
2.7.2	Fast RCNN Model	25
2.7.3	Faster RCNN Model	26
3.1.1	Flow of Faster RCNN	32
3.1.2	Inception-v1	33
3.1.3	Inception-v2	35
3.1.4	Loss on Training Set and Validation Set After Several Iterations	36
3.2	Labelling of 5 Different Components	38

4.1.1	Classification Loss and Localization Loss	40
4.1.2	Localization Loss and Objectness Loss	41
4.1.3	Total Loss and Clone Loss	42
4.2.1.1	Yellow Led	43
4.2.1.2	Red Led	43
4.2.1.3	White Led	43
4.2.1.4	Led X	43
4.2.1.5	Mini Led	43
4.2.2.1	Mini Led with Larger Scale	43
4.2.2.2	Mini Led with Smaller Scale	43
4.2.3.1	Two Mini Led	44
4.2.3.2	Led X and Mini Led	44
4.2.3.3	Yellow Led and White Led	44
4.2.3.4	Yellow Led and Red Led	44
4.2.3.5	Yellow Led and Mini Led	44
4.2.4	White Led in Bright Condition and Dim Condition	45
4.2.5.1	White Led	45
4.2.5.2	White Led	45
4.2.5.3	White Led	45
4.2.5.4	Red Led	45
4.2.5.5	Red Led	45
4.2.5.6	Red Led	45
4.2.5.7	Yellow Led	46
4.2.5.8	Yellow Led	46
4.2.5.9	Yellow Led	46

4.2.6.1	Led X and Mini Led	46
4.2.6.2	Yellow Led and Mini Led	47
4.2.7.1	Images of Yellow Led Classified Wrongly into White Led	47
4.2.7.2	Preprocessed Yellow Led	47
4.2.7.3	Preprocessed White Led	47
4.2.8	White Led with Two Boundary Box in Two Different Location	48
4.2.9	Yellow Led which has been Wrongly Localized	48
4.3	Unrecognized Object	49
4.4.1	Led X AP Graph	50
4.4.2	Mini Led AP Graph	50
4.4.3	Red Led AP Graph	50
4.4.4	White Led AP Graph	50
4.4.5	Yellow Led AP Graph	50
4.5	Performance Score of GPUs	54
4.6	Number of Images Used versus Mean Average Precision (%)	55

**LIST OF APPENDICES**

<b>APPENDIX</b>	<b>TITLE</b>	<b>PAGE</b>
A	Coding	64

## CHAPTER 1

### INTRODUCTION

#### 1.1 Background

Artificial intelligence (AI) is a machine system that consist of sense of human minds. It is define as an intelligent device which react according to its environment. When a machine is able to do some reasoning, self-correction and self-learning, it is known as an artificial intelligence system.

The birth of AI is during the year of 1956's at Dartmouth where Allen Newell, Herbert A.Simon and Cliff Shaw came out with the Logic Theorist (Brunette et al., 2009). They had initiate a series of research projects which related on the programming of computers to have human behaviours. The Logic Theorist was a software program that used to prove the theorems in symbolic logic. It is the first working program that able to simulate and solve some complex problem which require some aspect of human sense. This Logic Theorist and some other cognitive simulations developed by Allen Newell, Herbert A.Simon and Cliff Shaw had brought lots of opportunities for researcher to explore on the information-processing psychology developing field (Brunette et al., 2009). For current development, when dealing with some complex tasks studies in human factors psychology, Logic Theorist is still the central part of theory cognitive psychology, and the ideas from the Logic Theorist is still required for the use to solve those complex tasks.

During the late 1990s, AI start to taking into concern in the real world applicability. During the year of 1997, IBM's Deep Blue chess program had successfully defeat the world champion, Garry Kasparov and this is the time that prove that the AI system is getting more intelligent (May, 2017). Some real world application such as image recognition and speech recognition is start being developed by the researchers. Researches are finding way to allow the algorithms learn the logical rules by themselves without structuring the logical rules which set by humans manually (May, 2017). The researchers then shift their focus into the Artificial Neural Networks (ANNs). The ANNs was first invented during the year 1940s to mimic how the human brains learn. After a few decades, this ANNs was been use widely as the concept of the backpropagation of gradient descent was improved quite a lot (May, 2017). The backpropagation method helps to reduce the number of permutations required. Hence, it become more efficient when comes to the training session. But there are still some limitations with current technology that had plagued their adoption even with the use of improved new algorithms.

In 2006, some changes were made on the ANNs and now it is replaced by the term of deep learning neural networks (DNNs) which implemented by Geoffrey Hinton (May, 2017). Hinton added multiple layers of neural networks into ANNs in order to optimize the results obtained by each layer. Hence, there is a significant improve while the learning was now accumulated faster up the stack of layers. When the Graphical Processing Units (GPUs) was implemented, Andrew Ng of Stanford University make an improvement for the deep neural networks using the GPUs at year of 2012 (May, 2017). GPUs consist of massive parallel architecture that are able to handle multiple tasks simultaneously. Ng found that the training time used to train the deep learning neural networks is significantly reduced compared with the use of general purpose CPUs (May, 2017).

Currently, there are 4 main factors which help to drive the AI today. First of it is the "Big Data" was introduced. AI require a huge amount of data in order to learn more effectively and precisely. Big Data provide lots of data information from different sources such as mobile computing, Internet of Things sensors and social. The second factor is about the cheap computation power. Hardware is the one which remained as a constrain factor after the AI algorithms was improved at the past. Now, GPU gained



the popularity in the AI community as it provides high computational power which can run the operations parallel and perform matrix multiplication in a more efficient manner. Not only the implementation of GPU, CPUs have also been improved as it now able to perform more efficient matrix computation and more effective parallelization with the new deep learning instruction set implement in the CPUs processor (May, 2017). The third factor is a more sophisticated algorithms was implemented in recent. The state of the art in AI is depending on how the algorithms works. Hence, a more advanced algorithm would allow the AI to solve some of the specific problems such as speech recognition and image classification with a more precise accuracy. Lastly, broader investment is also one of the main factor to drive the AI towards the future. In the past, the funding in this area is not enough combined with some challenging problems met in AI resulted in minimal progress. Now, AI investment is keeping increase as many people see the possibility and the benefits of AI which able to makes lots of working tasks become simple.

## **1.2 Problem Statements**

Deep learning might be the best way in training a better AI but when it comes to training process, it definitely requires lots of data to make sure it would achieve a more precise accuracy. Just like a normal human brain, a lots of real world experiences must be acquired in order to learn and deduce from it. For the artificial neural networks, it requires a huge amount of data to abstract more parameters in more details. For example, an image object recognition tasks require at least thousands of training data image in order to allow the machine extract and recognize the details of the object clearly.

In the previous decade, object detection is hard to be implement because of the insufficient of dataset and the lack of powerful CPU resources. After the implementation of GPU by NVIDIA in year 1999, developing a deep learning model is not a dream anymore. The time used for training and testing the dataset had been reduced significantly. A powerful GPU can at least decrease 70% of the training and testing time for a particular model. Many researchers start to do research and working

on different type of architecture for the convolutional neural network. The accuracy of a classifier is getting better and better as the design of architecture used for the convolutional neural networks are keep improving. The amount of data used would also affect the quality of a classifier model. Insufficient data would lead to bad performance.

With current technology, we now could obtain data image easily from different type of resources and the data can also be transfer easily through high speed internet. But yet for specific item, it might hard to be found on the internet. Hence, we still need to capture our own image if we want to use it as our dataset. A classifier is just available to classify an image into one category of classes. For example, in an image which consist of a cat, a classifier would identify it as a cat. If an image contains more than one objects, the classifier will also available to categories it into a certain class only. Object detection cannot be built using a standard convolutional neural network followed by a fully-connected layer. This is because the number of occurrences of the object of interest will vary for different input image. An object detection system requires the use of deep learning technique which involves implementation of several neural networks for both image localization and classification. If an input image consists of few objects, a boundary box would be drawn around each object and then it will input those boxes of image into a neural network for classifications.

Overfitting in neural networks is the problem that will occur in many of the cases. Overfitting is basically when a neural network unable to learn in an effective manner due to several causes (O'Shea and Nash, 2015). Besides, overfitting can also be explained in the case when the training accuracy is increasing but the validation accuracy is decreasing. It most probably would occur in the complex models which having huge amount of parameters relative to the number of observations. The efficiency of a model is not just only judged by its performance on the training dataset that fed in but also its ability to perform well on an unseen data set. The model would only recognize those training data model but does not learn how to generalize to new situations and data set.

In deep learning for an image classifier, optimizing the hyperparameter's value is not an easy task. The value of a hyperparameters is define at the prior to the commencement of the learning process. A small change in the hyperparameters value

would invoke a large change in the performance of the model. The hyperparameters value which relying on the default parameters value without any optimization would bring a large impact on the model performance. Learning rate, number of epochs, batch size, activation function, weight initialization, and dropout layers are the hyperparameters in a convolutional neural networks. All these parameters value would affect the accuracy of outputs. There are two type of methods in order to find out the best suit hyperparameter's value which is the grid search and the randomized search. Grid search would search all the parameter combinations for given values. Parameter space of a given number of candidates is generated by random search with a specified distribution. For grid search, it is more commonly used in many machine learning of algorithms but not for deep learning algorithms. Random search is more suitable for tuning the deep learning neural network's hyperparameter and it is more efficient compare to grid search.

### **1.3 Applications**

A very well-known application of object detection used in is the self-driving cars to detect objects around. As the industrial now is moving into 4.0 industrial which means fully automation industrial, this deep learning technique can also be implemented for this industrial use to reduce the need of human works. For example, this object detection application can be used in is the inspection process to detect the defect of a wafer product or component.

Over the years, many of the semiconductor manufacturers had done lots of efforts to reduce the time used for each process and also reduce the need of human workers to operate each process. All of these can be a huge expense for a semiconductor company. Hence, for the past few years an automatic optical inspection machine has been developed. It was mainly to tackle the problem of inspecting defect products which take a long time and require many workers. Before the deep learning methods was used, there are several techniques such as image processing and machine learning is used to solve that inspection problem. The accuracy of image processing is very low compare to machine learning. For the image processing, its orientation is quite

important. Any mis-orientation would lead to the system to classify it as defect. For the machine learning, it requires engineers manually hand-featured its specifications. It takes time to specify all the feature of a component and its accuracy turn out is still acceptable. Lightning is also an important issue that need to be worked out for both machine learning and image processing technique in developing auto-inspection machine. A slight change of lighting sources would affect the output of the results.

#### **1.4 Aims and Objectives**

The below shows the objectives of the thesis:

- i) To study the use convolutional neural network for object detection purpose.
- ii) To build a deep learning based image component detection using Tensorflow framework in Python language.
- iii) To classify and localize the electronic components using the object detection model.
- iv) To detect and classify components in real-time.

#### **1.5 Scope**

The assignment entails the development of deep learning model running on the tensorflow gpu which capable to identify and classify the components accurately in real time form. The below shown few challenges that had been met for this project:

1. Gathering and labelling of the training and testing data.
2. Classifying components which has almost similar appearance but which actually are two different components.
3. Identifying of multiple components in a single frame where some components might be only partially visible or overlapping with the others.

4. For the whole process, it need to be done really fast and must be accurate. Hence, real time video detection is recommended. Besides, it also requires a very high specification GPU to run for the testing for fast performance.

## CHAPTER 2

### LITERATURE REVIEW

#### 2.1 Introductions

##### 2.1.1 Automated optical inspection (AOI) Machine

Automated optical inspection (AOI) Machine has been widely used in the manufacturing production line for quality control and real-time detection of defects. One of the example of defect structure is the damages on the backside of wafer due to the inaccuracy of dicing saw process. The main damage is the chipping and delamination of the backside wafer which can reduce the die strength (Mei et al., 2012). There are kinds of product such as wafer, semiconductor and led parts that are required this technology to enhance the speed and accuracy of the defect detection. Furthermore, it can reduce the needs of human supervision to take care of this inspection process.

Artificial neural networks have been applied on these AOI machines in order ensure the accuracy for defect detection is great enough. But there are still some of the defect features such as the flaw size and inclination might not be recognized (Ye et al., 2018). Hence, a more intelligent classification system has to be developed to evaluate different kinds of defect features. The development of AI technology must be improved in order to improve the feature-based defect classification. In the previous, the defect classification approaches are based on the classic neural networks which most of the time require the programmer to specify the characteristic of the features without the ability of self-learning (Ye et al., 2018). As technology getting greater and greater,

deep learning methods was now introduced and used in the AOI system to output a more precise accuracy.

## 2.2 Machine Learning, Artificial Intelligence and Deep Learning

Artificial intelligence (AI) is the intelligence acquired by a machine. It would know how to do some human tasks such as reasoning, learning, planning and communicating (May, 2017). AI is generally a branch of computer science which focused on developing machines capable of intelligent behaviour. For machine and deep learning, both of them using the algorithms to learn from given data and make prediction on another new set of data (May, 2017). The algorithm required a huge amount of data in order to perform well in specific task. Machine and deep learning is definitely the subset of the AI and deep learning is also the subset of the machine learning which focus even more narrowly on machine learning techniques that require “thought”.

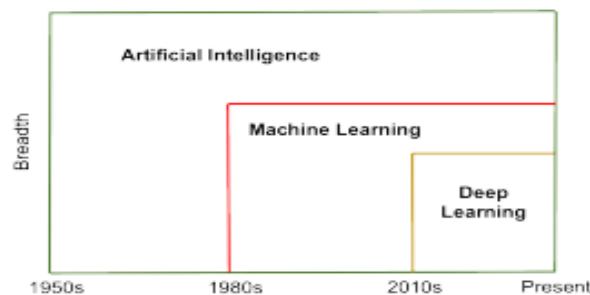


Figure 2.1: Relationship between Machine learning, Deep learning and Artificial Intelligence (May, 2017)

### 2.2.1 Machine Learning

According to Arthur Samuel in 1959, he mentioned that machine learning helps to reduce the work of programming needed for the learning process. Machine learning would learn from the data and make predictions on data through the explores study and

construction of algorithms. There are two type of machine learning methods are widely used in recent real world applications. One of the machine learning methods is the supervised learning and the another is the unsupervised learning method.

Supervised learning algorithms is given labelled samples and so the machine was trained to recognize the input and give a correct output based on the input given (Ongsulee, 2017). For examples, an image could be labelled either cat or dog and then when a new image was inserted, it would outcome a correct output based on the image given. The algorithm would compare the actual output with the correct outputs to find the errors and try to learn from it (Ongsulee, 2017). Supervised learning will use the pattern of a labelled input to predict the value on the unlabelled data. Supervised learning requires the supervision of a human and the labelled data was manually labelled by the programmer. There are 2 groups of supervised learning that it can be categorize which one is the regression and the another is the classification problem.

Unsupervised learning is a more interesting way of learning to study the representation of input patterns in a way that reflects the statistical structure of the overall patterns (Dayan, 2009). Unsupervised learning would react more likely to a human brain compared to the supervised learning. The training set used in unsupervised learning do not include any labels. The system was not told taught in recognizing the exact answer (Ongsulee, 2017). The algorithm must figure out by its own about what is being inputted. Unsupervised learning most of the time used in the transactional data and it is working quite well on it (Ongsulee, 2017).

Other than the supervised learning and the unsupervised learning, there are actually 2 more type of machine learning exist which is the semi-supervised learning and reinforcement learning. In semi-supervised learning, the word “semi” represent the need of the programmer in order to do correction whenever the machine did some mistake. It uses both unlabelled and labelled data for the training purposes. This method of learning is usually useful in the classification, regression and prediction process (Ongsulee, 2017). Whenever the cost associated is too high for labelling data, semi-supervised learning is recommended to be applied in order to reduce the cost by using more unlabelled data for training process (Ongsulee, 2017). For reinforcement learning, it basically learns through the trial and error process which would help to



optimize the action yield accuracy (Ongsulee, 2017). Gaming, robotics and navigation application would often consider to use this reinforcement learning method. Three primary components are required to take in consideration in this learning process which is the environment, the agent and the actions. The environment is referring to the things that the agent which is also known as the decision maker would interact with. Other than that, the actions are basically what the agent can do. The main objective is to optimize the expected outcome for a given period by choosing the right actions by the agent (Ongsulee, 2017).

### **2.2.2 Deep Learning**

Deep learning is a very powerful technique which is the enhancement of the machine learning. It is basically the integration of branch of machine learning. Deep learning has removed the need for feature engineering and replaced it with a brittle, complex and engineering heavy pipelines with one end to one end trainable models with the help of different tensor operations. Deep learning has the capability to learn from the past prediction by its own and to continually improve their predictions based on new testing data (Ongsulee, 2017).

Through this continuous learning process, it would have the capability to detect different kinds of defect structure even though the defect structure is small and tiny. Thus, the deep neural networks were constructed and inputted into the AOI system for a better defect classification and evaluation (Ongsulee, 2017). A graphics processing unit (GPU) card was required to deal with huge amount of training set data for the classification of defect structures for different kinds of electronic components. A faster training process could be done with the help of GPUs.

## **2.3 Convolutional Neural Network Architecture**

### **2.3.1 LeNet (Created by LeCun)**

LeNet was the first convolutional neural networks that built in 1990s which start the field in deep learning. It was mainly used for hand-written number or character recognition.

LeNet-5 was the latest architecture version of LeNet which is implemented at 1998. LeNet-5 containing 7 layers with different trainable parameters also known as the weights (Patrick et al, 1998). In this case, the size of input image used is larger than the largest character size in the database. The size basically set at 32x32pixel which is larger than the largest character size of 20x20 pixels at most. This is to ensure the end-points or corner features would be detected. LeNet-5 used normalization for the input pixels so that the learning rate would be faster (Patrick et al., 1998). The layers of the LeNet-5 are arranged accordingly where the first layer is the convolutional layer, followed by a sub-sampling layer, again a convolutional layer and a sub-sampling layer, once again a convolutional layer, then a fully connected layer and finally an output layer. The process for recognizing higher resolution images might require more convolutional layers. Hence, LeNet architecture is more suitable in classifying hand-written numbers or character's due to its resources was limited.

### **2.3.2 AlexNet**

AlexNet is almost similar with the LeNet architecture. AlexNet has a deeper network compare to LeNet where there is an increase in the filters per layer and also with the stacked convolutional layer. It consisted of convolutional layer, dropout layer, max pooling layer and also the fully connected layer. ReLU activations function was performed at the end of each convolutional layer and fully connected layer. The introduced “dropout” layer was used to set the output of each hidden neuron to zero or exclude it when it probability achieved is below 0.5 (Krizhevsky and Geoffrey, 2012). This can allow the network to learn more robust features which are useful. Those

unused random subsets of the other neurons are ignored in its backpropagation process to reduce the time consuming for the whole neural networks to complete its process.

### **2.3.3 ZFNet (Created by Zeiler and Fergus)**

ZFNet model is a model architecture which modified from the AlexNet model architecture to have a more interesting way in visualizing feature maps. ZFNet used a smaller filter size and with a decreased stride value in order to retain more information for the original pixel. A larger filter and a larger stride value used in AlexNet might loss some of the relevant information especially during the process in the first convolutional layer. In this ZFNet, it has introduced a new visualization technique which is the Deconvolutional Network. It able to examine the different feature activations and the relation of its compare with the input pixels (Zeiler and Fergus, 2014). Deconvolutional networks work in an opposite way of what a convolutional network does. The input will go through a series of unpool, rectify, and filter operations for each preceding layer till it reaches the input space. Through these processes, the type of structures excited could be examine for a given feature map (Zeiler and Fergus, 2014).

### **2.3.4 GoogLeNet/Inception**

Inception is used to differentiate some scale of invariance in an object recognition. A convent might not be able to recognize an object if the object is too small in an image. The network filters of the Inception would enlarge the object and to be trained to recognize it. Inception has the capability to convolve through an image using various filter sizes which both process at same computational step in the network (Murphy, 2016).

At the end of the GoogLeNet network, it does not consist of a fully connected layer. It is replaced by an average pooling process across a  $7 \times 7 \times 512$  output volume in

order to yield  $1 \times 1 \times 512$  vector (Murphy, 2016). The purpose of replacing the fully connected layer to an average pooling layer is to avoid the substantial portion of parameters required for fully connected layers. In constructing a high level representation with low dimension of an image, the successive recombination of high level parameters in a fully connected layers is excessive (Murphy, 2016).

### **2.3.5 VGGNet (Created by Visual Geometry Group)**

VGG-16 consists of 16 layers which is a very uniform architecture. It is similar to the AlexNet architecture but with lots of filters used for extracting more parameters and a deeper weight layers where AlexNet only consists of 8 weight layers (Yu et al., 2016). With the huge amount of filters used in the convolutional layer, this would cause it to have over millions of parameters and would take a very long time for its training process. VGG-16 is with better representation ability and also good in removing those unrelated background information (Yu et al., 2016). It is more preferable to be used in those challenging baseline feature extractor. Its network architecture weights are quite large due to its depth and number of fully connected nodes is very high.

### **2.3.6 ResNet (Residual Neural Network)**

During the ILSVRC competition in 2015, Microsoft ResNet won that competition. The ResNet have the solution for the backpropagation encountering the exploding or vanishing gradient problem (Murphy, 2016). The output of the network is depends on the sum of a series of small residual changes applied on the original input. This network has the possibility to perform even more deeper than anything previously possible. At the competition, Microsoft has successfully implemented a residual network which having 152 layers which is equivalent to 8times deeper compared the VGGNet (Murphy, 2016). For the previous network design, increase in network depth would actually decrease the network performance. But this is different when it come to the design

architecture of a residual network. ResNet performance continues to increase as the depth increase.

## **2.4 Basic Building Block of Convolutional Neural Network**

There are four main layers which represent the basic functionality of a convolutional neural network. First of all is the input layer which help to convert and resize an image into the matrix form of pixel values. Next, the convolutional layer which would determine the output of neurons through the calculations of the scalar product between the region of neurons connected to the inputs and also their weights. Rectifier linear unit is used in this layer which act as the activation function for the convolutional layers. Pooling layer will then perform the subsampling along the spatial dimensionality of a given input, it would only take the important parameters and reduce all other unused features. Max-pooling is the most common pooling process used in for the image recognition process. After that, the fully-connected layers was introduced and the ReLU activation function was applied in this layer just like the convolutional layer did. This layers are used to allow a high level reasoning to be processed and to produce the output for the classification purpose. All these operations are the basic building blocks of a convolutional neural network and also in any other different convolutional neural network architecture. All of these operations have their own important role in the convolutional neural networks.

### **2.4.1 Convolutional Layer**

The convolution operation is the main core building block of the convolutional network which undergoing most of the high computational tasks. It is used to extract features information from the input image. The convolutional layer would have a learnable filters extends through the full depth of the input volume. Assume an input image which having only 1 depth which is a grayscale image, a typical filter which have a size  $3 \times 3 \times 1$  would slide over the width and height of the input and

produce a 2-dimensional feature map. The number of feature map produced would depends on the amount of filters used in a convolutional layer. For example, if 16 filters were used in a convolutional layer, it will produce 16 features map which will stack together along the depth dimension. The features map can also be known as the activation maps which would give responses of that filter at every spatial position. The figure 2.3.1 is an example of input image which is in the array form. When a learnable filter shown in figure 2.3.2 slide across it from the top left corner until the bottom right corner, it will produce a feature map just like what shown in figure 2.3.3.

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Figure 2.2.1: Input Image in Array Form (Britz, 2015)

1	0	1
0	1	0
1	0	1

Figure 2.2.2: Learnable Filter with Certain Weight (Britz, 2015)

4	3	4
2	4	3
2	3	4

Figure 2.2.3: Convolved Feature Map (Britz, 2015)

#### 2.4.2 Adaptive activation function

Tanh, sigmoid or rectifier linear unit function in nonlinearity components could help to solve the complex function in a large neural network (Hornik et al., 1989). However,

the activation function chosen for the deep neural networks must be very careful as it consists of large impact to the network performance.

Activation function is one of the training hyperparameter in the deep neural networks. For the past few years, researcher has been developing the activation functions and make a great improvement on the activation chosen which help to speeds up the training process and to get more precise results. Adaptive piece-wise linear function is introduced during the year of 2014 as it could be learned for each hidden neuron (He et al., 2015). Then, a parametric form was proposed and those extremely deep models to be trained and achieve a high accuracy results on the GoogLeNet dataset (Zhang et al., 2015). Polynomial activation function which is smooth piece wise is then proposed to reduce the bias of the regression networks. Piece-wise linear function always assigns the negative part into zero. A predefined value would be exponentially decreased into zero when using the elu function whereas leaky relu function consist of small predefined negative slope for the negative inputs (Clevert et al., 2016). elu activation function able to accelerate the training process by preventing a bias shift to occur in the relu networks. A mixed function of elu and leaky relu is then proposed as an adaptive function which learn in a data-driven way (Qian et al., 2018).

Inspired from the above explored method, it had given contribution to the study of effect of asymmetry on a parametric sigmoid. The main objective is to produce an adaptive sigmoid function with parameter which can be fitted and controlled for each of the hidden neuron with the other deep neural network parameters in the training process. Besides, one of the other aim is to propose a smooth variant relu function feed into the deep learning neural networks. These activation functions which is in parametric form has improved the effectiveness of the deep neural network in the classification task such as image and text classification (Farhadi, 2017).

In recent, there are two model of classification functions is used in the convolutional neural network which is the softmax function and the Rectified Linear Units (ReLU) function. Softmax function is basically used in the last layer of the neural network. Discrete probability distribution for the K classes is described by the softmax function. ReLU is an activation function which has been proven by strong mathematical and biological analysis for the neural network activation purpose (Agarap,

2019). It works by thresholding values into 0. For example,  $f(x) = \max(0, x)$  would output a value of 0 when  $x < 0$  and output a linear function when  $x \geq 0$ . The figure 2.4 below shows the ReLU function for  $f(x) = \max(0, x)$ .

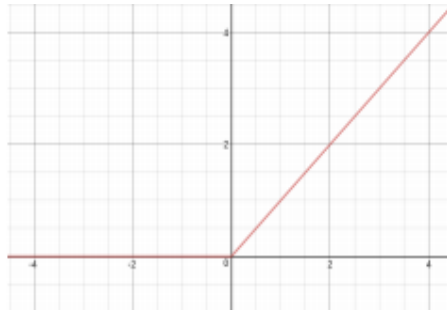


Figure 2.3: ReLU activation function for  $f(x) = \max(0, x)$  (Agarap, 2019)

ReLU is commonly used in the neural network as an activation function whereas the softmax function is being used as the classification function for the last layer. Then, the softmax cross-entropy function is introduced to learn the weight parameters of the neural networks (Agarap, 2019). The weight parameters are learned using the backpropagation methods for the gradients from the ReLU classifier.

### 2.4.3 Max Pooling Layer

After passing through the convolutional layer, a max pooling layer is then applied for the subsampling purpose. The max pooling layer is mainly to reduce the spatial size of the input, number of parameters and the computations to prevent overfitting. It is basically applied on a non-overlapping slide of the receptive fields with certain value of stride to slide through the corresponding input. The zero padding parameter might need to be set in this process to produce an output of equivalent dimensions. Zero padding is a process of padding around the border of the input and is an effective way to control the dimensionality of the output volumes (O'Shea and Nash, 2015). The down-sampled feature maps produced by the last max pooling layer are often follow up by the fully connected layer subsequently (Witten et al., 2016).



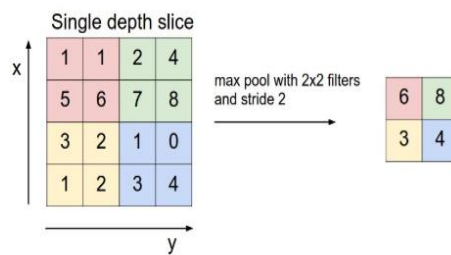


Figure 2.4: Max Pooling Process (Britz, 2015)

#### 2.4.4 Dropout

Dropout is an efficient method on reducing the training time for a huge neural network which might take up to several days of training period. Dropout is a method which remove or ignore those hidden neurons when the particular neuron achieves or fall below a certain amount of probability (Krizhevsky et al., 2012). The removed neurons would not contribute to the forward pass or participating in the backpropagation process. Each time an input is obtained, the neural network would sample in different architecture where all these architectures are sharing their weights. This help to reduce the complex co-adaptations of neurons since a neuron unable to rely on the presence of particular neurons. Hence, it has the capability to learn more robust features which are useful in conjunction with many other random subsets of neurons (Krizhevsky et al., 2012). Dropout layer is implemented at the fully connected layers which consist of the hidden neurons and involved in the backpropagation process.

#### 2.4.5 Optimization Algorithms

Different optimization algorithms used in deep learning neural networks would get different unexpected results based on its problems and datasets given. Optimization algorithms is used to update and calculate appropriate and optimize the value for such

model's parameters which would affect the model's learning process and the output of the model.

Gradient Descent is a first order optimization algorithm. First order optimization algorithm minimizes or maximizes a loss function using its gradient values relative to its parameter. A line which is tangential to a point on its problematic surface are basically a first order derivative. This technique is easier to compute and able to do fast converging on a huge data sets. As we know, backpropagation is a common technique which used to train a neural network. In this process, it will first propagate forward through the hidden layers and calculating the dot product of inputs and their corresponding weights. Then, activation function is applied to those sum of products and introduces non-linearities to the Model which allow the model to learn more functional mappings. After that, it will propagate backwards in the network and carrying the error terms and update the weights values using this gradient descent optimization method. Gradient descent will help to calculate the gradient of error function with respect to the weights and update the weight parameter in the opposite direction of the gradient of the loss function with respect to the Model's parameters (Ruder, 2016). The weight value would be finalized when the local minima point is found.

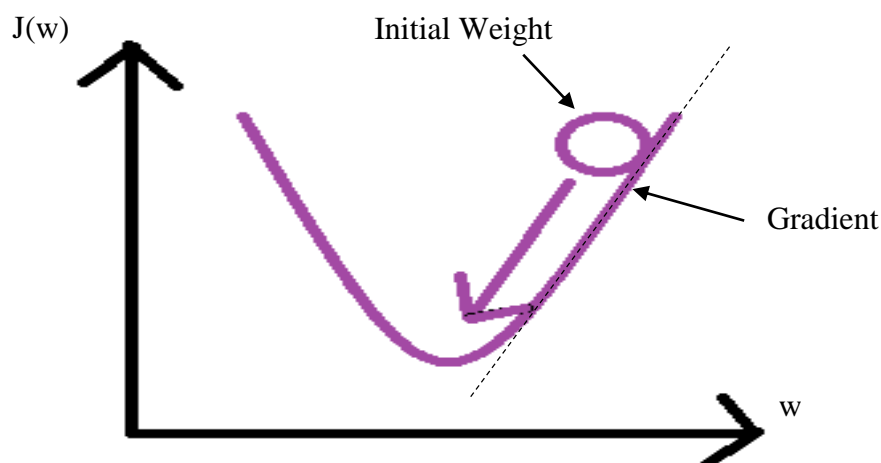


Figure 2.5: Gradient Descent Model

Stochastic gradient descent (SGD) help to update the parameter for each training example. Before this, gradient descent performs redundant computation for large datasets through recomputing the gradients for similar examples before each parameter was updated (Ruder, 2016). SGD do not follow this redundancy method but it performing one update at a time and it is a much faster way. The loss function is fluctuated to different intensities due to its frequent updates with high variance. Thus, there is a possibility to discover better local minima through these fluctuations (Ruder, 2016).

## **2.5 Object Detection Methods**

There has been a lot of research done in object recognition using the old computer vision method. One of the old method is by using the sliding window detector to detect the objects. Different size of window is used in order to find the location of object. After that, the feature is extracted using Histogram of Oriented Gradients, HOG feature extraction method and continued by using SVM classifier to classify it. This method is computationally expensive and it is very slow as it need different scale of windows and to slide it step by step. The accuracy for this method is significantly lower compare to the deep learning based methods. The deep learning based methods are divided into two categories which one is two stage detection method and the other is the unified detection method. Two stage detection methods consist of RCNN, Fast RCNN and Faster RCNN methods and for the unified detection method, it consists of YOLO and SSD methods. There are a few major concepts that are used in both of these techniques and it will be explained in the following sub-title.

### **2.5.1 Bounding Box**

A bounding box is a rectangular or a square box which is drawn to fit a particular object in. It consists of 4 components which is the coordinates of the left bottom, right bottom, upper left and upper right of the boxes. A bounding box is generated whenever it found

the object of interest. The predicted bounding box would be compared with the ground truth bounding box to compute the value of intersection over union. A ground truth box is a box which is labelled in the testing dataset by the user its own. Hence, the system can be trained to generate a bounding box through the optimization process between labelled bounding box and predicted bounding box.

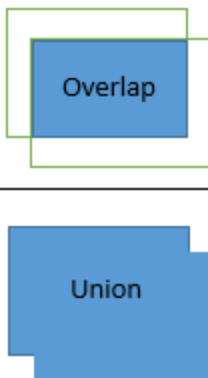
$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} = \frac{\text{Overlap}}{\text{Union}}$$


Figure 2.6.1: Computing the Intersection of Union

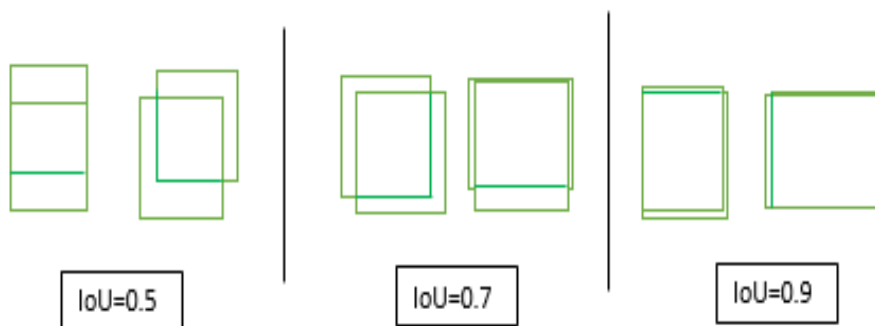


Figure 2.6.2: Example of Computed Intersection of Union

An IoU is used to find the best region box for the object of interest only. The higher the value of calculated IoU, the predicted box generated is more precise and located nicely around the object. A few examples of computed IoU are shown in Figure 2.6.2 above.

### 2.5.2 Anchor Boxes

An anchor has multiple scaling and each of it are with different aspects ratio. For example, assume it has 3 different type of scaling, and each of it consists of 3 different aspects ratio. Hence, the anchor boxes that has been generated are equal to 9 for each location. An example image of anchor boxes is shown in Figure 2.6.3 below.

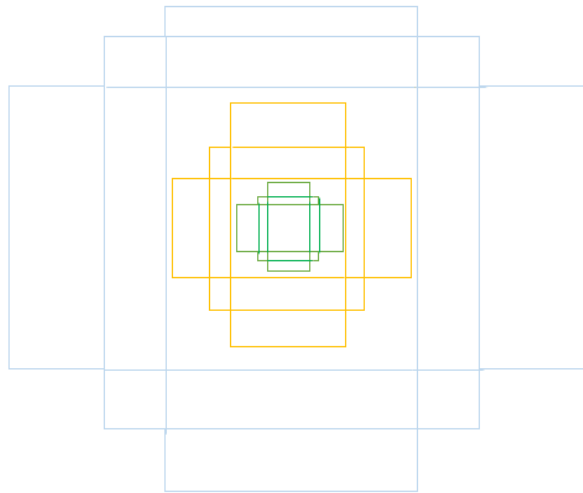


Figure 2.6.3: Anchor boxes

For  $y_{rpn\_cls}$  (Region Proposal Network Classification Layer), each anchor box consists of two values which defined as  $y_{is\_box\_valid}$  and  $y_{rpn\_overlap}$ . If  $y_{is\_box\_valid}$  is equal to 1 means the box generated is valid. If  $y_{rpn\_overlap}$  is equal to 1 means the box consist of an object (Xu, 2018). These values are used to calculate the losses for the classification. For  $y_{rpn\_regr}$  (Region Proposal Network Regression Layer), each anchor has four values which defined as  $tx$ ,  $ty$ ,  $tx_2$ ,  $ty_2$  and each of these consist of 2 values which is  $y_{is\_box\_valid}$  and  $y_{rpn\_overlap}$  (Xu, 2018). These values are used to calculate the losses of a boundary box and to refine it.

In short, if the IoU of the generated boundary box obtained is greater than 0.5, the particular anchor is known as foreground or a positive anchor. If IoU obtained is smaller than 0.1, the particular anchor is known as background or a negative anchor.

### **2.5.3 Classification and Regression**

The predicted bounding box is done using the regression methods. This regression process is to eliminate those duplicate boxes. Normally, a threshold is set for the IoU value in order to eliminate those inaccurate predicted bounding boxes. After the bounding box is obtained accurately, the classifier now would learn and recognize the features within the bounding box. During the classification process, it consists its own backpropagation process through the convolutional neural network of deep learning model or to optimize the losses for a particular model.

## **2.6 Two-Stage Method**

### **2.6.1 RCNN (Regional-Based Convolutional Neural Network)**

RCNN method is proposed by Ross Girshick et al. where this method used selective search to extract only 2000 regions of interest from an image. Selective search uses local cues like intensity color, texture and measure of insideness to identify the region of interest. In this RCNN, selective search was done by generating initial sub-segmentation in order to have many candidate regions. Then, by using the greedy algorithm to recursively combine those similar regions into one. Next, the generated regions are used to produce the final candidate region of proposals. These candidate regions are fed into a CNN to continue with the feature extraction process. Lastly, by using the SVM method it would classify the extracted regions into certain classes. The flow diagram of RCNN is shown in Figure 2.7.1 below. There are several disadvantages using this RCNN method for object detection. First of all, the time used for training is huge as it need to classify those 2000 regions proposal in an image. Besides, it is similar for testing, it also requires long time to complete its testing process. Hence, it is not suitable to use in real-time application. Selective search is a fixed algorithm. It does not consist of any learning process. It might lead to generating bad candidate regions.

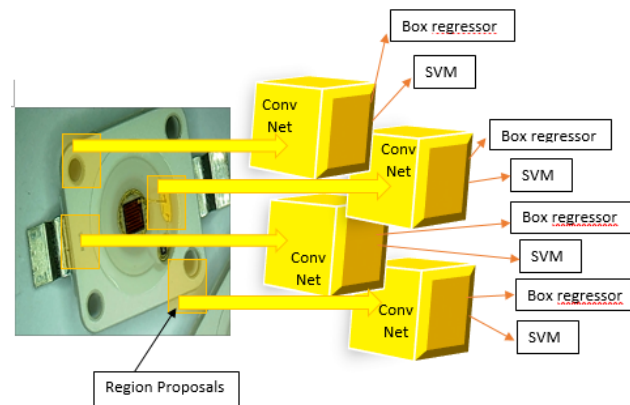


Figure 2.7.1: RCNN Model

## 2.6.2 Fast RCNN (Fast Regional-Based Convolutional Neural Network)

Fast RCNN methods is almost similar with the approach of RCNN. The difference between Fast RCNN and RCNN is just only we change the sequence of inputting image to CNN. For the Fast RCNN approach, it feed the image into the CNN first to generate the convolution feature map. Then, it just identifies the region of proposals using selective search. Fast RCNN consist of an additional ROI Pooling layer to reshape the region of proposals into a fixed size. Lastly, it fed into a fully connected layer, most probably would be the softmax layer to perform the classification process. Refinement process for the bounding boxes will also be performed. The overflow of diagram for Fast RCNN are shown in Figure 2.7.2 below. Compare to RCNN, Fast RCNN has the benefit of the convolution is done only once per image rather than those 2000 region of proposal which will take a long time to run.

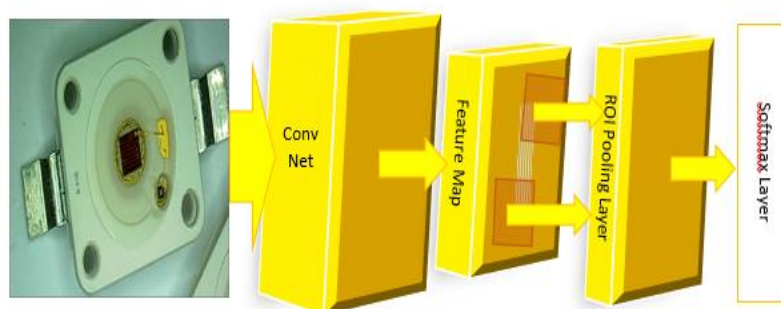


Figure 2.7.2: Fast RCNN Model

### 2.6.3 Faster RCNN (Faster Regional-Based Convolutional Neural Network)

Faster RCNN is an improved method of previous Fast RCNN and RCNN. Faster RCNN does not use selective search method to find those candidate regions. Selective search method is a slow and time consuming process. Faster RCNN allows the network to learn how to find the region proposals. The figure 2.7.3 below shows the process in Faster RCNN method.



Figure 2.7.3: Faster RCNN Model

Similar to Fast RCNN, image was first inputted into a CNN first to generate the feature map. Later, the generated convolution feature map will be into a region proposal network. Region proposal network would find the region of interest but there is no feature extracted. In the ROI Pooling layer, the region of interest found would match with the feature map. It will be resizing into a fixed shape. The resized region of interest would be input into a fully connected layer for classification. The bounding box is then refined in order to eliminate duplicated boxes.



## **2.7 Unified Method**

For the previous two-stage method require the producing of region proposals. For this unified method, it pre-define a set of boxes to look for objects.

### **2.7.1 YOLO (You Only Look Once)**

You Only Look Once (YOLO) is a method which look at the entire image. Image was split into  $S \times S$  grid. Each grid would contain number of  $M$  bounding boxes. For each bounding box, it has its own class probability. Single convolutional neural network would predict the bounding boxes and the class probabilities for all these boxes. If the bounding boxes confidence score and class probability is above the value of threshold set, then the bounding box will be use to locate the object. YOLO only predict one type of class in each grid. Hence, it would be struggling in detecting a small object or very close object.

### **2.7.2 SSD (Single Shot Detector)**

SSD achieve good balance between the speed and accuracy performance. SSD used auxiliary convolutional layers to extract features at multiple scales. These extracted feature maps will be input into a convolutional kernel to predict the bounding box and classification probability. The score of the class probability and 4 offset (coordinates of bounding box) is computed. Those score which exceed the threshold point would be use as the final box to locate the object.

## **2.8 Comparison of SSD with Faster RCNN**

Based on the research of Phon-Amnuaisuk et.al, they compared the method of SSD and Faster RCNN in detecting the appearance of baby which are less than 12months old inside a house. They find out that SSD is able to predict the bounding box accurately but Faster RCNN turn out has a better classification accuracy. Faster RCNN has an accuracy of 97.5% whereas SSD is only 86.1% accurate in classifying the image of a baby (Phon-Amnuaisuk et al., 2018). In this project, we require a better classification accuracy rather than focusing on the bounding box accuracy. Hence, we would use the Faster RCNN network for our research purpose in classifying different components.

## **2.9 Preferred Coding Language**

Python is the preferred language in building project which involving artificial intelligence, machine learning or deep learning process. Python is an object-oriented programming based, high level and interpreted programming language (Mandl, 2012). One of the benefits of using Python is that involve less coding. Python able to implement the same logic using only one fifth code as compared to other object-oriented programming language. Python also contain prebuilt libraries. There are lots of libraries such as Numpy, Tensorflow and matplotlib for the need in AI project. All these prebuilt libraries help the programmer to save their time in writing coding (Mandl, 2012). Besides, python is also an open source application with a great community. There are lots of related examples of different type of AI projects script available in the online webpage. It consist of a huge community which are active all the time and willing to help the programmers whenever they meet difficulties in their projects. There is quite a large amount of Python developers outside the world. With the extended libraries and active community, Python are getting more popular and becoming the hottest language for kinds of projects in today's world.

## CHAPTER 3

### METHODOLOGY

#### 3.1 Approach

This section explains the procedure and rationality behind the conducted experiments and the choice of method used. The goal of this experiments was to access the performance of the object detection algorithms and to evaluate their ability in recognizing components. The network used in this experiments are Faster RCNN which is widely favoured.

#### 3.2 Data gathering and labelling

Image data is collected by using microscope and webcam. The need of microscope to capture the image is because of the size of the components is very small. The image captured are in the format of jpg format and it is in the size of 600x480 resolutions. There are 5 different components are used in this experiment and it needed to be classified out later into different categories. About a total of 250 images was captured for all these different components which placed in different locations or backgrounds.

A software named “labellmg” is used for image annotation. The tool enables the user to easily draw a bounding box to annotate the objects with a pre-defined label. The saved annotation was an .xml file. This .xml file consist of the label data and the

coordination of the object for each image. In an image, it might consist more than one annotations.

### **3.3 Hardware and Software**

The software required in this experiment is the Python software and anaconda software. Python was chosen for the development for this project as it is easier to use and the source example could be obtained easily from the webserver. The choices of framework used would be the Tensorflow framework for deep learning. Tensorflow is one of the famous framework which consist of prebuilt libraries inside the python software and can be used to design, build and train for deep learning models. The training process for the networks was completed using following hardware specification: GPU- NVIDIA GeForce 840M (2GB), CPU- Intel i7-5500U @ 2.40GHz, RAM- 8GB, Operating System- Window 10. A camera and a microscope is required to obtain the inspection images. Lighting is provided at both sides of the lens to get a clearer and constant image illumination.

### **3.4 Model Training Details**

The networks model used would be explained in this section.

#### **3.4.1 Faster RCNN**

The model of object detection used in this project is based on Faster RCNN method. The process flow for this architecture is shown in figure 3.1.1 below. Inside a Faster RCNN, it would run through 2 different networks which is a region proposal network and a feature extractor network. A region proposal network is used to identify the candidate regions of an object in an image whereas a feature extractor network is used to extract the features of an image.

Faster RCNN would first process the input image with the feature extractor network, which is the inception-v2 network in this case. This inception-v2 consists of a convolution layer and pooling layer to extract the feature maps. It is then passed to the RPN network to obtain the region proposals. RPN is sharing the full-image convolution feature map with the detection network. It is designed as a fully-convolutional network which has the ability to predict object bounding boxes and the object score at each position (Zhao et.al, 2018). RPN would scan over the feature maps to find out the object in an image is either background or foreground. The object present in an image might be various in size and shapes. Hence, multiple  $k$  windows which is defined as anchor boxes would be used to identify the location of the object. Each region proposed would consist of an “objectness” score and 4 coordinates to represent the bounding box for that region. Regions proposed by the RPN network might be redundant due to its proposal of multiple regions for the same object. Non-maximum suppression is used in this case to solve the problem of redundancy. The regions of proposals are set to a maximum number of 300 to enhance the efficiency and effectiveness during training. A ROI pooling layer was introduced because the generated region of proposals are various in size. ROI pooling layer would help to convert the output of RPN into fixed-sized vector and to be input to a fully connected layer. The fully connected layer used in this project was the softmax layer. Refine for bounding box position would be done in order to get the most actual bounding box for each object in the image.

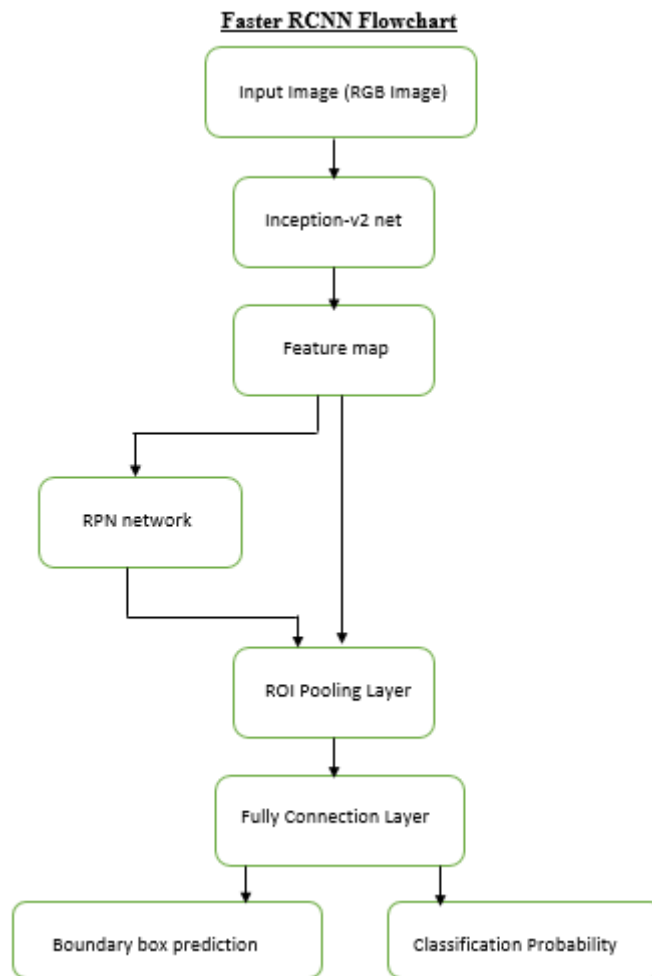


Figure 3.1.1: Flow of Faster RCNN

### 3.4.2 Inception Network

Inception-V2 act as a feature extractor. Feature at certain selected intermediate layer of this network are used to find the class-agnostic box proposals. These box proposals are then used to crop features which is in the same intermediate feature map. After the matching of the box proposals with the cropping of feature map, it is then fed into the fully connected layers to predict the categories of class it fall in. Box refinement need to be done for each region proposals in order to obtain the actual object location.

Inception network is a heavily engineered network. The main reason to develop this network is to optimize the performance of an CNN in term of speed and accuracy.

In the inception-v1, it used to solve the problem of the large variation of object size in an image. An object in an image could be small or large in size but there have similar properties. The differences in area of an object occupied by the object in an image could cause the choosing of right kernel size for convolution operation become difficult. Large kernel is used for information that is distributed more globally and a small kernel is used for information that is distributed more locally.

Inception-v1 introduce multiple sizes of filter to operate on the same level. Hence, this makes the network to become wider rather than deeper. If a network is getting deeper, it might be prone to overfitting and the might facing with the problem of vanishing gradient. Inception-v1 is implemented in the intermediate layer of a convolutional neural network. The input of inception-v1 might passes through lots of filter and cause the inputs to become “deep”. When a neural networks become deep, the particular neural network is computationally expensive. Hence, in this inception-v1 the author limit the number of input channels by adding in an extra operation 1x1 convolutions before passing through the multiple sizes filter as shown in Figure 3.1.2 (Raj, 2018). Auxiliary classifiers are also introduced by the author to prevent the gradients from the middle part to be vanished out.

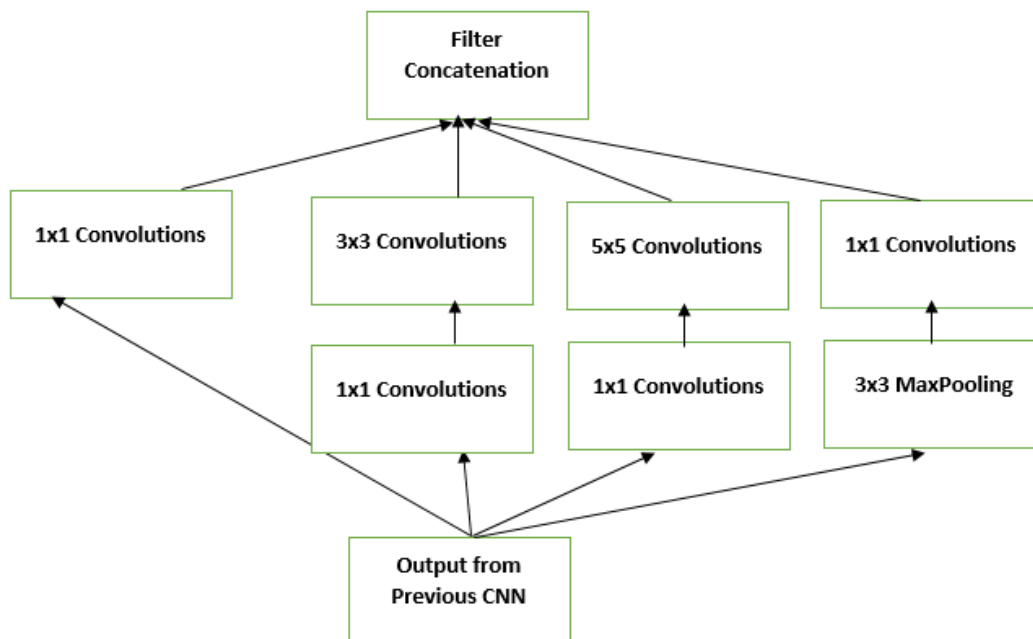


Figure 3.1.2: Inception-v1

Inception-v2 is implemented to reduce the computational complexity of the previous method. Neural networks perform better when convolutions did not make changes on the dimensions of the input drastically. If ones reduce the dimension too much, it might cause loss of information. This issue also known as a “representational bottleneck”. In this inception-v2, the author introduces smart factorization methods so that the convolutions process is much more efficient when computing its complexity (Phon-Amnuasisuk et al., 2018). For example, a 5x5 convolution would be replace by two 3x3 convolution operations which give the same outputs which is 1.38 time cheaper. Moreover, the 3x3 convolutions can be replace with the combination of 1x3 and 3x1 convolutions. The diagram of smart factorization for inception-v2 can be seen in Figure 3.1.3. Combination of 1x3 and 3x1 consist of only 6 parameters and 3x3 consist of 9 parameters. Hence, this method is 33% more cheaper than the single 3x3 convolutions. Besides, batch normalization is also introduced in this inception-v2 network to solve the problem of internal covariate shift. Covariate shift is refer to the change in the input distribution to a learning system. In a convolutional neural networks, parameter of all the input layers would affect the input produced to each different layer. Any small changes in the input would get amplified down the network (Shagunsodhani, 2019). The change in the input distribution to those internal layers of a deep neural network would occur. Batch normalization would help to normalize the output in each hidden layer. Batch normalization used the exponentially weight averages and bias correction to do the corrections. The use of weight and bias correction is to keep the mean and variance fixed. After normalizing, the scale of input features would still remain almost similar as original. This is some way of making those layer to be independent and would not affect by the other layers.



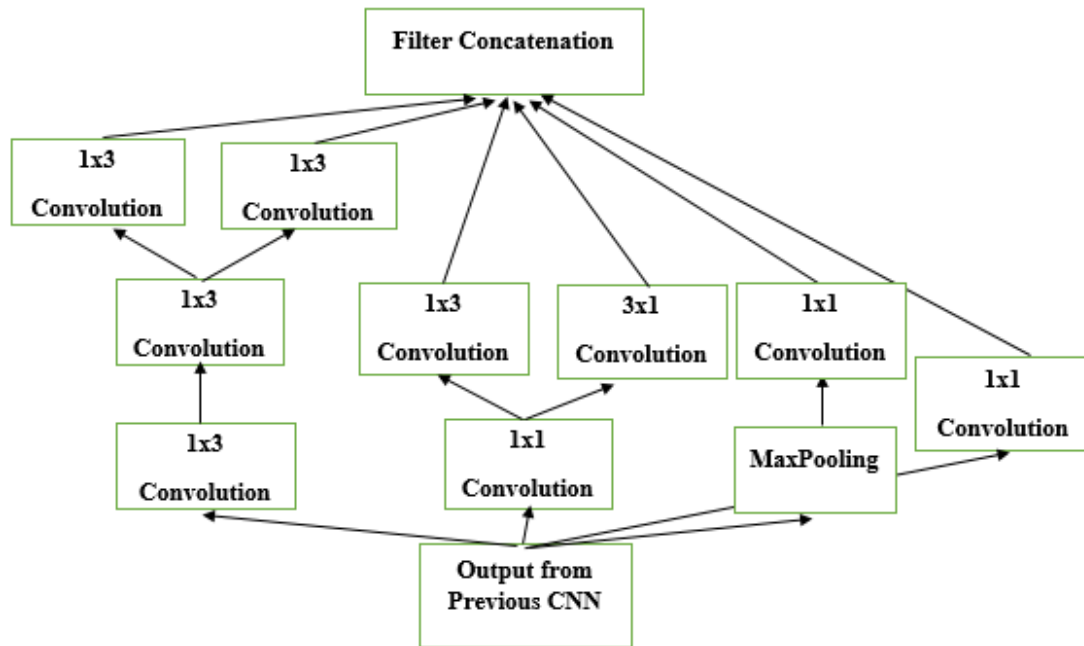


Figure 3.1.3: Inception-v2

### 3.4.3 Network Setting

In this project, the input image inserted is a RGB image that has been resized into 224x224 dimension. This network is on its best performance with the input image resolution between 600 and 1024. The object would be classified into 5 different classes. There are 4 different scaling of anchor boxes were being generated with a ratio of 0.25, 0.5, 1 and 2. For each scales, it consists of 3 different aspect ratio. The aspect ratios used are 0.5, 1 and 2. The regularizer used in the first stage for the box predictor is the L2 regularizer. Through regularization, we could prevent the overfitting to occur by penalizing complex models. Overfitting occurs when the generalization curve shows the loss of training loss is decreasing but the validation loss eventually goes up. An example of generalization curve which meet with this situation are shown in Figure 3.1.4 below.

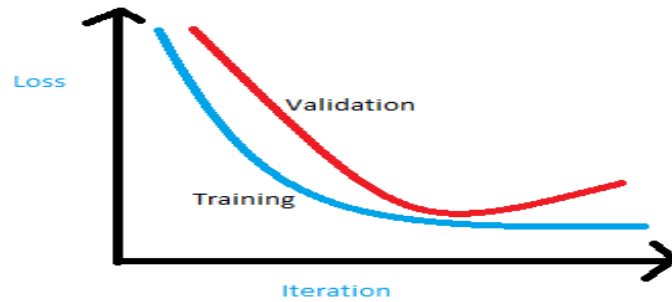


Figure 3.1.4: Loss on Training Set and Validation Set After Several Iterations

The loss term measures only how well the model fits the training data. With this regularization term, it would help to measure the complexity of the model. Now we have all these measurement, we can minimize the losses and also the complexity for a particular model. The model complexity must be minimized because it represents the weights of all the features in the model and the total number of features with non-zero weights. During region proposal, the initializer used is the truncated normal initializer with a standard deviation of 0.01. An initializer is used to initialize the value for the weight and bias. This truncated normal initializer would generate random values except those values with more than two standard deviations from the mean. The non-max suppression for the IoU threshold value is set at 0.7 which means only those predicted boxes with IoU threshold value more than 0.7 just will be remained and the maximum amount of box proposals is 300. The batch size is set at 1 as the GPU resources is limited. Mask RCNN is used to extends Faster RCNN to a pixel-level image segmentation. In Faster RCNN, it consists of third branch to predict the object mask which is in parallel with the existing branches for localization and classification (Weng, 2017). This mask branch is applied on a RoI pooling layer to predict the segmentation mask in the form of pixel to pixel manner. In pixel level segmentation, a much more fine-grained alignment is required. Mask RCNN help to improve the RoI pooling layer so that the RoI can be more accurate and precise when mapping with the regions of the original image. Mask RCNN is used to fix those locations with misalignment cause during the quantization or resizing process in the RoI pooling layer. The learning rate was at first 0.0002 and then later when reach till 90k step it would decrease to 0.00002 which is 10time smaller than the initial setting. Learning rate is a hyperparameter which controls the adjustment for the weights of the network with respect to the loss gradient.

The larger the value of learning rate, the faster the training would proceed and the losses would decrease significantly. But the large value of learning rate might not optimize the losses in an ideal way. It might be keep fluctuating and make us could not hit the ideal target loss. Low value of learning rate might travel slower along the slope of the losses curve but it ensures that each local minima point could be achieved. It might take a long time to travel along the slope but it is worth to wait for the favorable outcome. To fasten the process, we avoid from using fixed value for the learning rate. We use a larger initial value at first which is consider as a suitable value for the initial setting. Later, we just decrease the initial learning rate to a smaller value so it can optimize more further.

During the classification of validation image, the network would go through the forward propagation step and output a probability for each class. The output probability is depending on the weights which have been optimized correctly during the training process. The more convolution layers we have, the more complicated features could be obtained. Example of complicated features are edges and scratches. The neurons in the last fully connected layer represent the number of classes that we have for a particular model. This fully connected layer are also known as the output layer.

### **3.5 Experiment Setup**

Due to the CPU resources is insufficient to run the CNN, we have to run this neural networks with a GPU. To access the GPU, CUDA v9.0 and cuDNN v7 must be downloaded. Besides, some other software like Anaconda and Python is downloaded to run the coding. The Tensorflow Object Detection API repository is downloaded from GitHub and a directory folder is setup. The model for Faster-RCNN-Inception-v2 is obtained from the official Tensorflow's model zoo and it is placed in the same directory folder.

An Anaconda virtual environment is setup specifically for this experiment. First, we have to install the tensorflow-gpu v1.10.0 and the python version used must set to v3.5.2 Only this version could help to support the use of GPU on windows. Some

additional libraries such as anaconda, protobuf, pillow, lxml, Cython, jupyter, matplotlib, pandas and opencv-python are required to be downloaded in order to run the full coding. The PYTHONPATH must be setup and point to the required directories. Protobufs files which are needed by Tensorflow to configure model and training parameters are compiled and setup.

Once we had completed our environment setup, then we start gather the pictures of different components. In this experiment, 5 different components are needed to be classified which is White Led, Red Led, Yellow Led, Led X, and Mini Led as shown in Figure 3.41 below. To train a good detection classifier, the images captures should have consisted the desired objects and some other random objects. Besides, the object captured should also come along with a variety of backgrounds and lightning conditions. Overlapped images or halfway images are also allowed to be used as training data. Next, use the software “Labellmg” to annotate the desired objects in each image that captured. The labelled annotation is saved in .xml file and all these .xml file is then converted into a .csv file. 80% of the images is used as training dataset and the remaining 20% is used as test dataset. Hence, two .csv files containing the data for the train and test images is generated which one named as train\_label.csv and the other is the test\_label.csv. These .csv file is later used to generate the record files for training dataset and testing dataset. These files are the main files that will be used to train the object detection classifier.

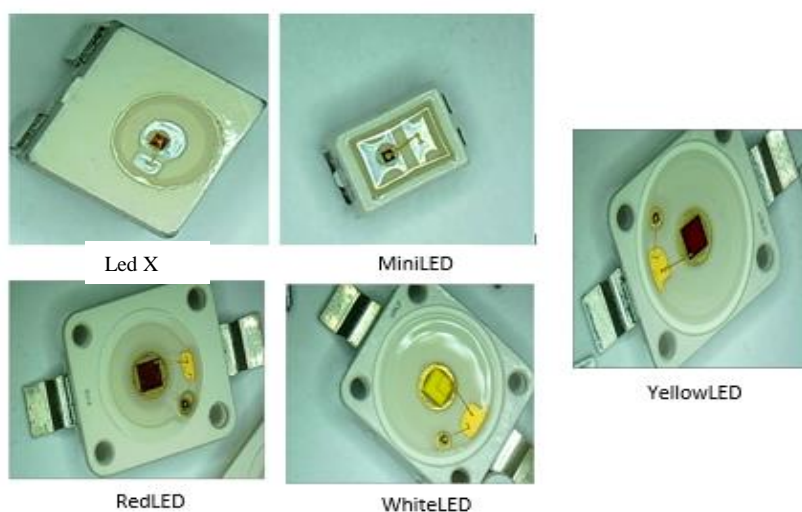


Figure 3.2: Labelling of 5 Different Components

A label map is created which tell the trainer what the components is defined by mapping it class name to it class ID numbers and save it as .pbtxt file. After that, configure the component detection training pipeline. When completed the configuration, we now can start our training process. During the training process, backpropagation method was applied in the convolutional neural layer in order to compute the difference between the actual outputs and the target. In between the hidden layers and the output layers, the algorithm will keep computing until such that the output result is close the target. After the training process was completed, generate the frozen inference graph by exporting it from the last step of training process. This frozen inference graph would be our model for classification. Now, we can test it out on new component images and check the accuracy for this particular component detection classifier.

### **3.6 Testing**

A total number of 50 images is used for testing in order to find out the mean average precision for the overall model. At first, the model was loaded by inputting the frozen inference graph to obtain the optimized weight and bias value which is set during the training. The component would be detected if only the score of the classifier and the boxes are higher than 80%. In other word, only things with confidence level which is higher than 80% would show the labelling score and the localization box around.

## CHAPTER 4

### RESULTS AND DISCUSSIONS

#### 4.1 Losses

##### 4.1.1 Box Classifier Loss

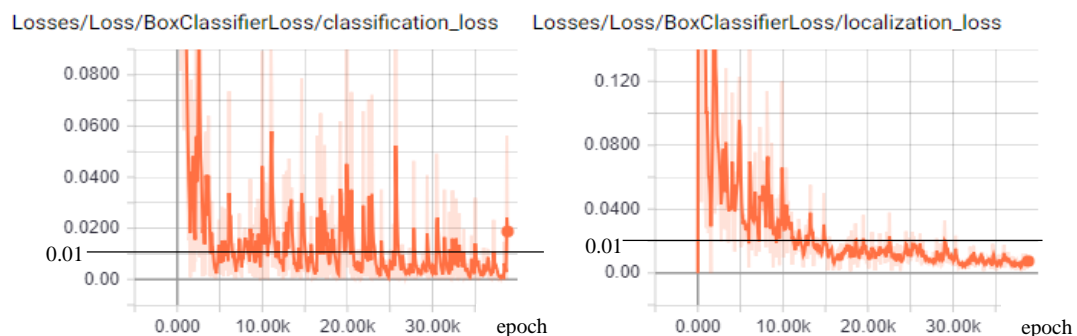


Figure 4.1.1: Classification Loss and Localization Loss. The graph is generated from the tensorboard.

For box classifier loss, it measures the losses of the boundary box predicted for both classification and localization. Classification loss is the loss for the classification of detected objects into various classes. In this case, the detected object is classified into 5 different classes. For the localization loss, it is referring to the loss of the boundary box regressor. It measures how accurate the box fit the object in. From the figure 4.1.1 above, we can see that the classification loss is decreasing very fast till the point 0.01 losses. It only took 5k epoch to reach the point while the localization loss took around 15k epoch to reach the point of 0.01 losses. This means that the classification layer is

learning faster compare to the regression layer. This situation occur because of the accuracy for objectness is already high and able to recognize what the object are but at the same time, the bounding box coordination still can't accurately localize the boundary box to fit nicely the object. Hence, the boundary box requires more time to learn for the localization prediction. A target line is set at 0.01 for both classification and localization losses which mean the training would only be stop after the model achieved an accuracy below this target line.

#### 4.1.2 RPN Loss

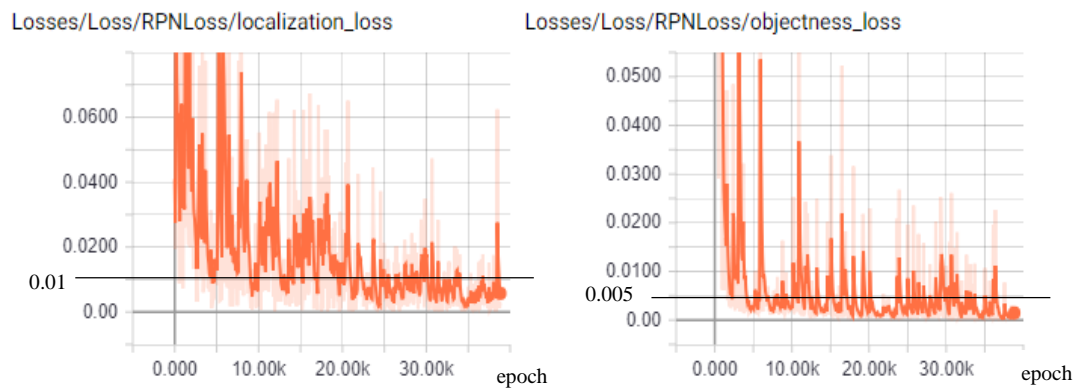


Figure 4.1.2: Localization Loss and Objectness Loss. The graph is generated from tensorboard.

For RPN loss, it measures the losses of the boundary box predicted during region proposals. Localization loss is still the same referring to the loss of the boundary box regressor but it is for the region proposals use. The objectness loss is the loss of the classifier that classifies if a boundary box is an object of interest. From the Figure 4.1.2, we can observe that objectness loss converge faster than the localization loss. The objectness loss reached at the point of 0.005 losses after passes through 5k epoch whereas the localization loss reached at the similar point of losses at 10k epoch. Similar as previous situation, to ensure the localization to propose accurately, it require more time to learn. A target line is set at 0.01 for the localization loss and 0.005 for the objectness loss.

### 4.1.3 Total Loss and Clone Loss

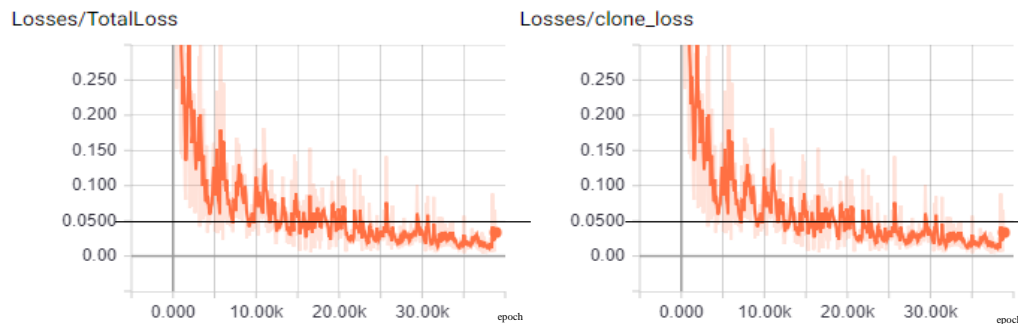


Figure 4.1.3: Total Loss and Clone Loss. The graph is generated from tensorboard.

Total loss is the losses for both RPN network and the Classifier network. In measure the average losses for all of the losses occur in RPN network and Classifier network as shown in figure 4.1.1 and figure 4.1.2. From figure 4.1.3, we can observe that the total loss is decreasing slowly which mean that the learning process is still occurring and the networks are trying to optimize their losses. The training should stop only after the losses reached the point of 0.05. In this case, the training is stop when the losses are around 0.02. The clone loss is similar as the total loss graph due to only using a single GPU. The clone loss would be different only if we train it on multiple GPUs. Then, the Tensorflow would create a clone of the model to train on each GPU and report the loss on each clone. This total loss does not represent the model can identify accurately when we use a totally new image for testing. Hence, validation dataset is input into this model for testing in order to find out its mean average precision (mAP). For each epoch, it took around 1.5seconds to complete using Nvidia GT840M GPU. To hit the target line which is set at 0.05, the number of epoch require is roughly 30k epochs which require 11hours to achieve the target. All the target line is set to ensure the best performance for a particular model could achieve.



## 4.2 Validation Data

### 4.2.1 Correct Detection

In this first part, we would first discuss about those images which has been identified accurately.

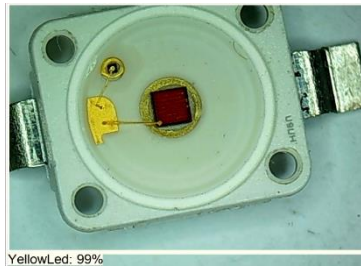


Figure 4.2.1.1: Yellow Led



Figure 4.2.1.2: Red Led

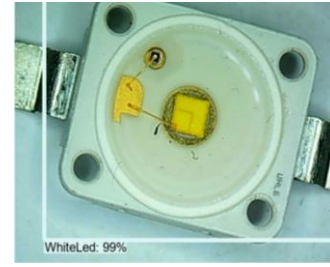


Figure 4.2.1.3: White Led

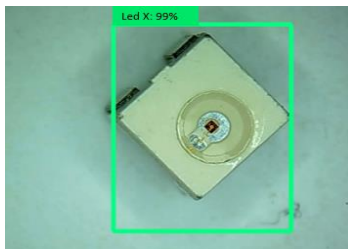


Figure 4.2.1.4: Led X

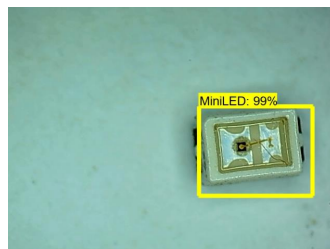


Figure 4.2.1.5: Mini Led

From the Figure 4.2.1.1-4.2.1.5 shows that this model able to predict all of the validation images accurately if the image consists only one component and the object captured is a full image of the object.

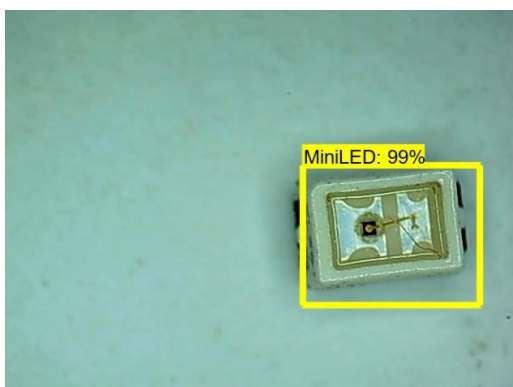


Figure 4.2.2.1: Mini Led with Larger Scale

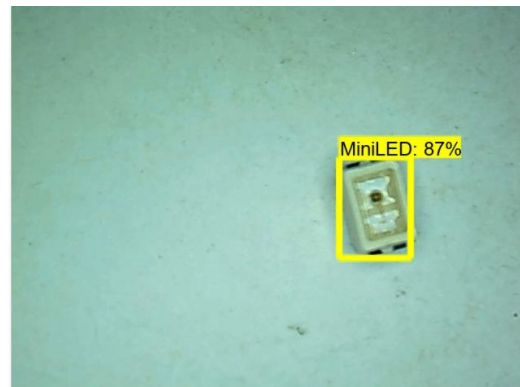


Figure 4.2.2.2: Mini Led with Smaller Scale

Image different scaling might cause the confidence level to decrease as show in Figure 4.2.2.1 and Figure 4.2.2.2. This is due to the image with smaller scaling might not be clear enough for the model to detect it.

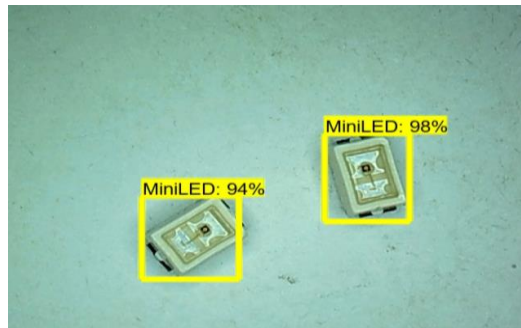


Figure 4.2.3.1: Two Mini Led

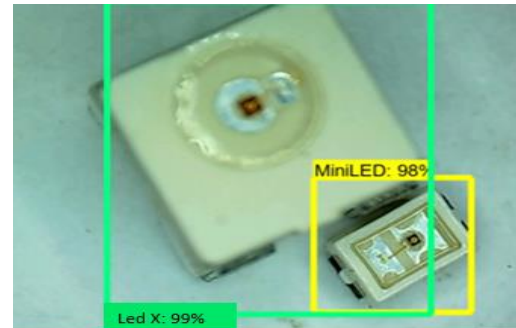


Figure 4.2.3.2: Led X and Mini Led

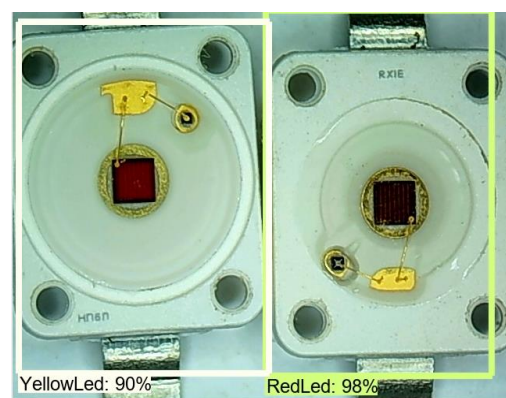
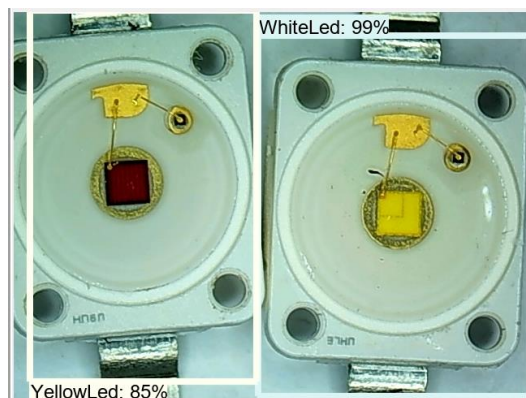


Figure 4.2.3.3: Yellow Led and White Led Figure 4.2.3.4: Yellow Led and Red Led

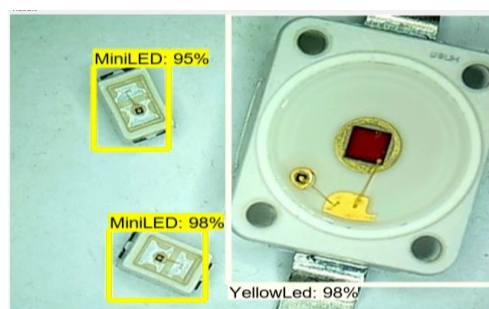


Figure 4.2.3.5: Yellow Led and Mini Led

From the Figure 4.2.3.1-4.2.3.5 shows the component which is side by side are also able to be identified and localized accurately. In Figure 4.2.3.2 shows that the microscope is focusing on the Mini Led and cause the Led X to become blur but it still able to identify it with a lower confidence score.

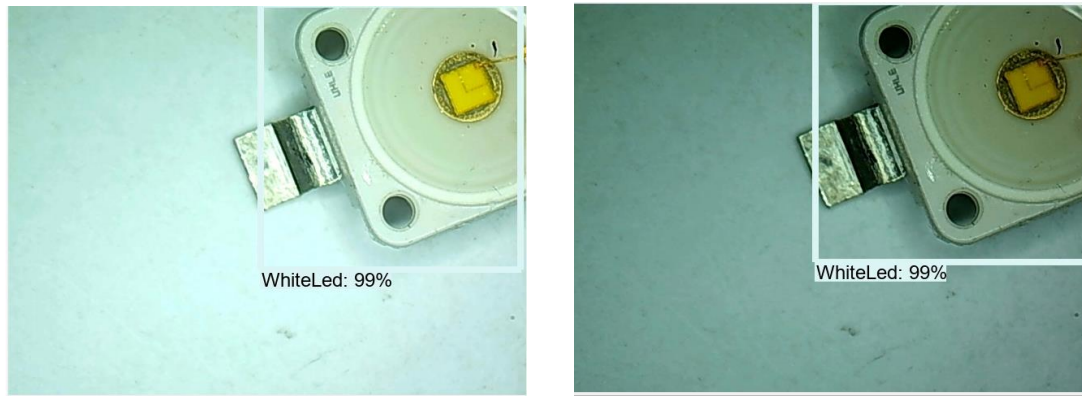


Figure 4.2.4: White Led in Bright Condition and Dim Condition

The lightning condition does not affect the output result of the object detection model. In the deep neural network several pre-processing have been done onto the image, feature extracted does not depend on the difference of lightning condition. Hence, the object detection is still able to identify it accurately even in a dim condition as show in figure 4.2.4.

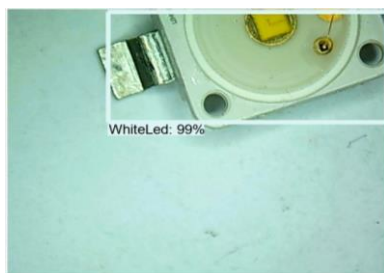


Figure 4.2.5.1: White Led

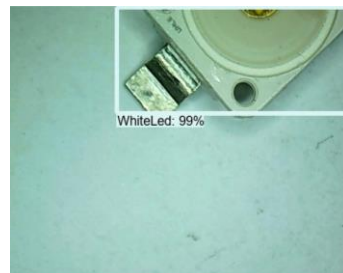


Figure 4.2.5.2: White Led

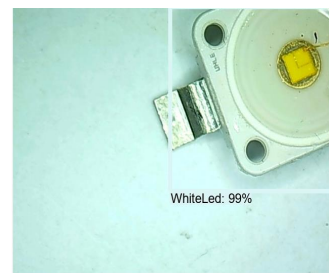


Figure 4.2.5.3: White Led

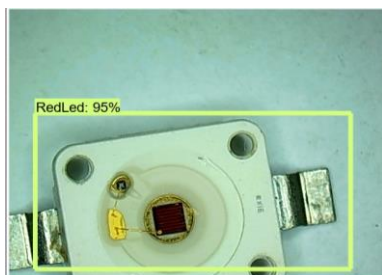


Figure 4.2.5.4: Red Led



Figure 4.2.5.5: Red Led

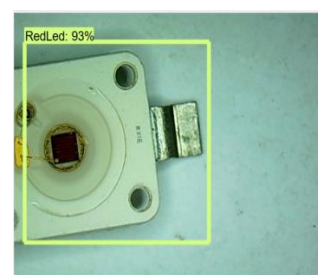


Figure 4.2.5.6: Red Led

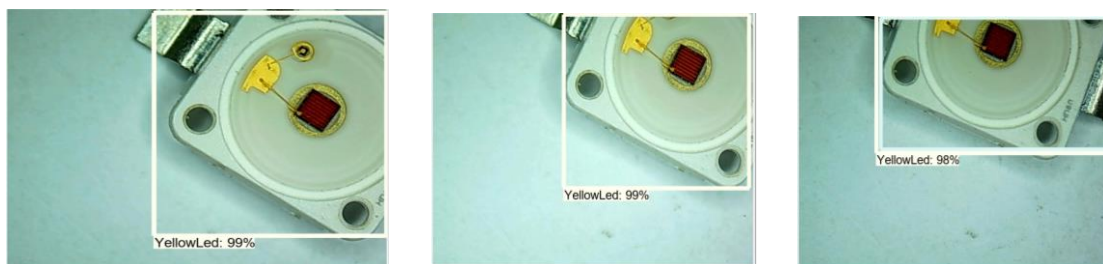


Figure 4.2.5.7: Yellow Led Figure 4.2.5.8: Yellow Led Figure 4.2.5.9: Yellow Led

From the figure 4.2.5.1-4.2.5.9 above, it shows different cropped image for White Led, Red Led and Yellow Led. The trained object detection model is still able to identify them accurately. This is because inside the training and testing dataset, we also provide of these cropped image which is similar as above. This is to teach them to recognize the components even if it is a halfway image which certain feature is still retained. In human perspective, once we see a thing very frequently, we can identify it even if it is a halfway image. Same as in this deep learning, once it identified certain feature which similar in their “memories”, it will compare with its “memories” and categories it into certain classes which it thinks it is most likely.

## 4.2.2 Error Detection

In this second part, we would discuss about those image with object not classified, has been classified wrongly or the boundary box is wrongly localized.



Figure 4.2.6.1: Led X and Mini Led

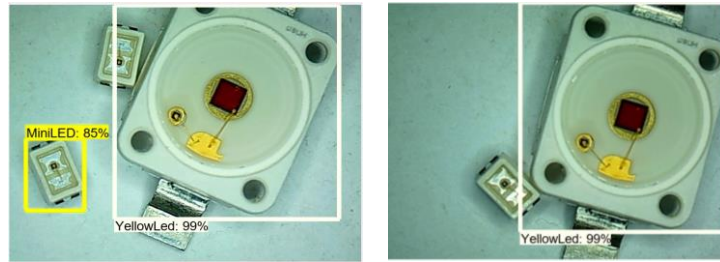


Figure 4.2.6.2: Yellow Led and Mini Led

From Figure 4.2.6.1 and 4.2.6.2 above, we could observe that the model somehow unable to identify all the object in the images. Smaller object or blurred object might easily be ignored by the model and cause the object to be unclassified. This problem is most likely can be solved if we input more images with different object inside it for the training process.



Figure 4.2.7.1: Images of Yellow Led Classified Wrongly into White Led

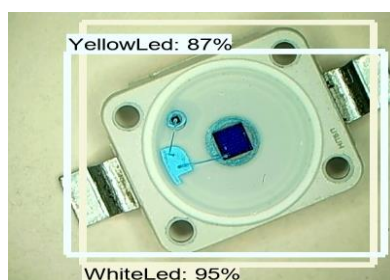


Figure 4.2.7.2: Preprocessed Yellow Led



Figure 4.2.7.3: Preprocessed White Led

From the Figure 4.2.7.1 above, it shows that the Yellow Led is classified wrongly into White Led. This situation occurs because the White Led has almost similar feature as Yellow Led if we do not include its color information. During testing, the input image would be preprocessed and the color information of the image might be washed out as

shown in Figure 4.2.7.2 and 4.2.7.3. The pattern difference between the Yellow Led and White Led is the center part of it. More details feature on the center part must be extracted for training session in order to improve the accuracy of classification. Different type of feature extractor network might help in solving this problem.

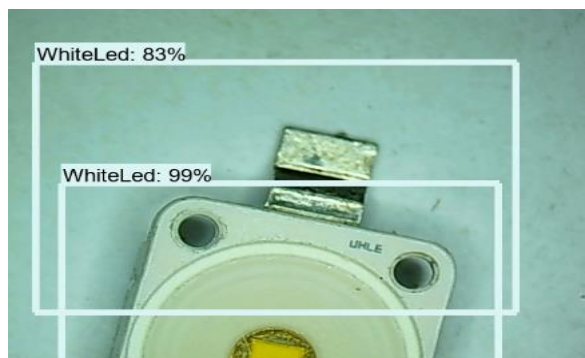


Figure 4.2.8: White Led with Two Boundary Box in Two Different Location

From Figure 4.2.8 above, two boundary box is generated on an images which consist only one White Led. This problem is due to input too many cropped images for White Led into the training session. As in previous, we had mentioned that by inputting cropped object images can allow us to identify it even if it is a halfway image. This would lead to this problem to occur. The object detection model would identify the object by part.

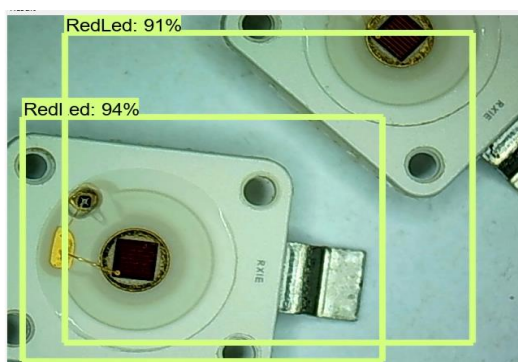


Figure 4.2.9: Yellow Led which has been Wrongly Localized

Figure 4.2.9 shows that one of the yellow Led has been wrongly localized in other location/ The boundary box tend to appear at inaccurate position. This problem almost

similar as the previous situation for the object not detected. It also can be solved by inputting more images with object side by side for training to improve its localization.

### 4.2.3 Unrecognized Data

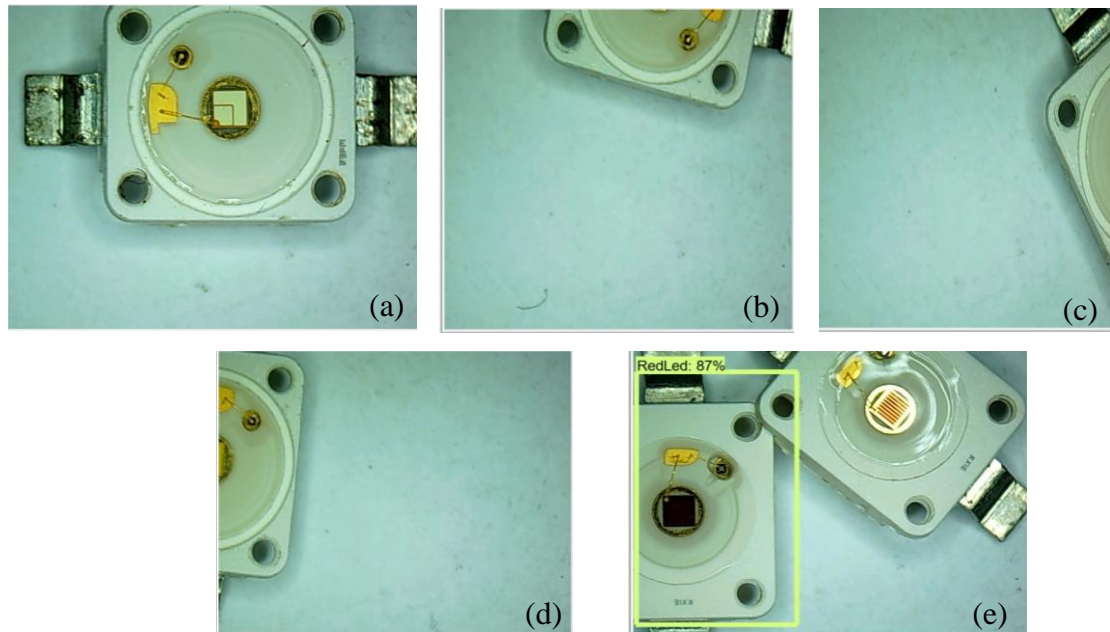


Figure 4.3: Unrecognized Object

In the Figure 4.3(a), it is a different component which we do not included in this experiment. Hence, the model would not classify it into any classes. For the figure 4.3(b)-4.3(d), the image does not provide enough feature information. Hence, the model also unable to categories them into any classes. From the Figure 4.3(e), it shows that one of the Red Led is overlapped with the other and cause the middle golden part of it have the reflection of light. This turn out that the Yellow Led consist of different feature information and so the object detection classifier is unable to classify it

## 4.3 Analysis

### 4.3.1 Mean Average Precision

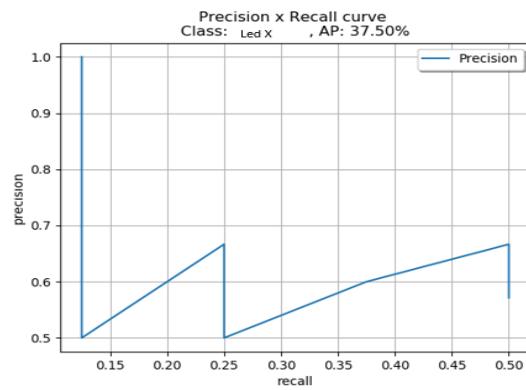


Figure 4.4.1: Led X AP Graph

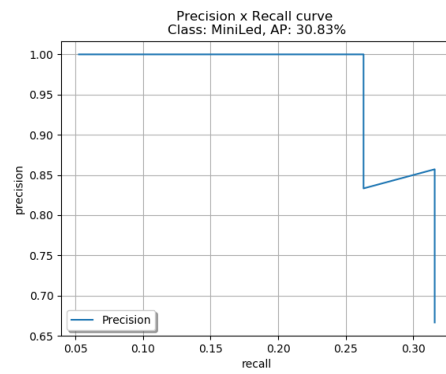


Figure 4.4.2: MiniLed AP Graph

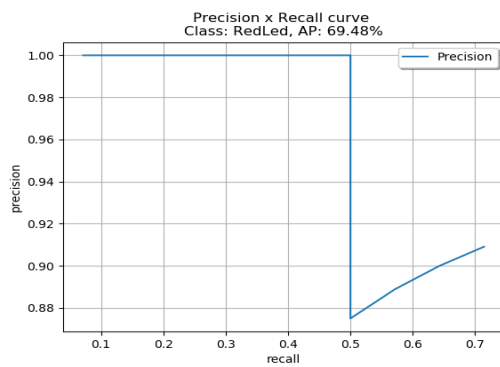


Figure 4.4.3: RedLed AP Graph

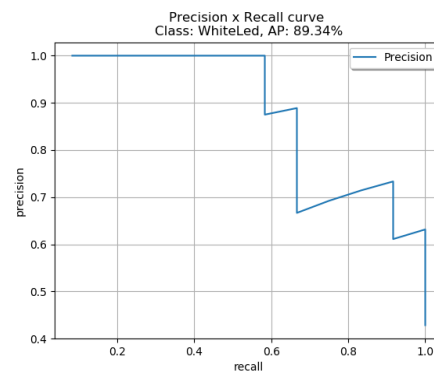


Figure 4.4.4: WhiteLed AP Graph

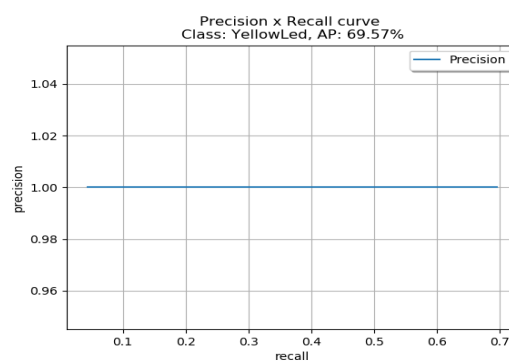


Figure 4.4.5: YellowLed AP Graph

Each of the graph from Figure 4.4.1-4.4.5 above show the average precision for each component. All of the graph are generated from Pattern Analysis, Statistical Modelling



and Computational Learning Visual Object Classes (PASCAL VOC) 2012. This graph would measure the trade-off between true positive rate and true positive predictive value for this deep learning model using different IoU thresholds. Precision measures the ability of a model to detect only relevant objects. Recall measures the ability of a model to identify all relevant cases. The IoU threshold here used is 0.5 which means those predicted object which exceed an IoU 0.5 would be classify as true positive. A true positive result indicate the result is a correct detection. A false positive result which below IoU threshold would be indicate as a wrong detection. IoU is the area overlapped between predicted bounding box and ground truth box. Object where ground truth was not detected is indicate as a false negative result. The precision is calculated through the equation 4.1 shown below:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \quad (4.1)$$

The Recall is computed through the equation 4.2 shown below:

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \quad (4.2)$$

If the false positive obtained is equal to 0, this means it has high precision whereas if the false negative obtained is equal to 0, this means that it has high recall. When the precision remains high as the recall increases, this mean that it would has a high average precision (Padilla, 2019). The graph computed is based on 50 different validation data.

The average precision obtained for each classes is shown in the Table 4 below. The mAP value would be the average value of the 5 classes and it is found to be 59.34%. The mAP value calculated are based on Pattern Analysis, Statistical Modelling and Computational Learning Visual Object Classes (PASCAL VOC) 2012 which the input variable consists of class name, left coordination, top coordination, width and height information. PASCAL VOC assist in providing evaluation performance for different object classes recognition.

**Table 4.1: Average Precision (in percentage) for each Different Classes**

Class	Average Precision(%)
Led X	37.50
Mini Led	30.83
Red Led	69.48
White Led	89.34
Yellow Led	69.57
<b>mAP</b>	<b>59.34</b>

### 4.3.2 Comparison of GPU Speed

**Table 4.2: Performance score for different type of GPUs**

GPU	Performance Score	Memory
Nvidia Titan RTX	100	24GB
Nvidia GeForce RTX 2080 Ti	98.4	11GB
Nvidia GeForce RTX 2080	96.1	8GB
Nvidia Titan X	96.0	12GB
Nvidia GeForce GTX 1080Ti	96.0	11GB
AMD Radeon VII	92.4	16GB
Nvidia GeForce RTX 2070	87.2	8GB
AMD Radeon RX Vega 64	84.4	8GB
Nvidia GeForce GTX 1080	84.3	8GB
Nvidia GeForce GTX 1070Ti	78.5	8GB
Nvidia GeForce RTX 2060	77.6	6GB
AMD Radeon RX Vega 56	76.7	8GB
Nvidia GeForce GTX 1660 Ti	71.4	8GB
Nvidia GeForce GTX 1050Ti	33.1	4GB
AMD Radeon RX 560	28.6	4GB
Nvidia GeForce GTX1050	28.1	2GB
AMD Radeon RX550	17.9	4GB
Nvidia GeForce GT1030	13.0	2GB
Nvidia GeForce GT840	11.0	2GB
Nvidia GeForce GT765	7.0	2GB

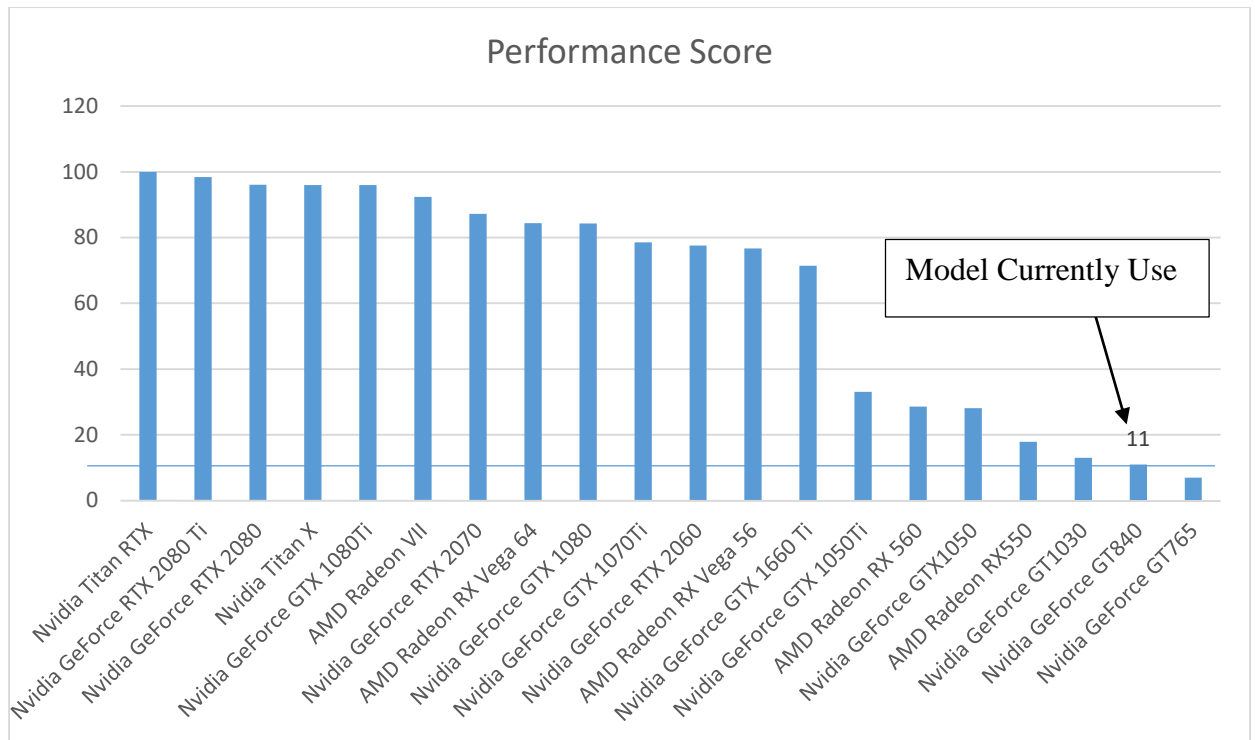


Figure 4.5: Performance Score of GPUs

The information in table 4.2 is obtained from a website resources (Angelini, 2019). The Figure 4.5 above show the performance score of different type of GPUs. The higher the performance score, the faster the performance speed of a GPU. The model which currently used in this project is only 11 performance score. It is consider having slow performance compare with the other advance GPUs.

**Table 4.3: Comparison of time used per epoch for training using different GPUs**

Type of GPU	Training Time Per Epoch(s)	Memory
Nvidia GeForce GTX1050Ti	0.3	4GB
Nvidia GeForce GT840	1.3	2GB

In the Table 4.3 above, we could see that the Nvidia GeForce GTX1050Ti has a faster performance speed compare to Nvidia GeForce GT840. From the given time shown in the table, we know that the Nvidia GeForce GTX1050Ti is about 4time faster than the GT840 as it consists of a larger memory space.

### 4.3.3 Comparison of Mean Average Precision using Different Amount of Images

**Table 4.4: Mean Average Precision Obtained using Different Amount of Images for Training**

Class \ No. of images	FasterRCNN-250images Average Precision (%)	FasterRCNN-150images Average Precision (%)	FasterRCNN-50images Average Precision (%)
Led X	37.5	6.32	4.33
MiniLed	30.83	40	25.8
RedLed	69.48	80.41	50.23
WhiteLed	89.34	51.62	40.5
YellowLed	69.57	25.71	10.6
<b>mAP</b>	<b>59.34</b>	<b>40.81</b>	<b>26.29</b>

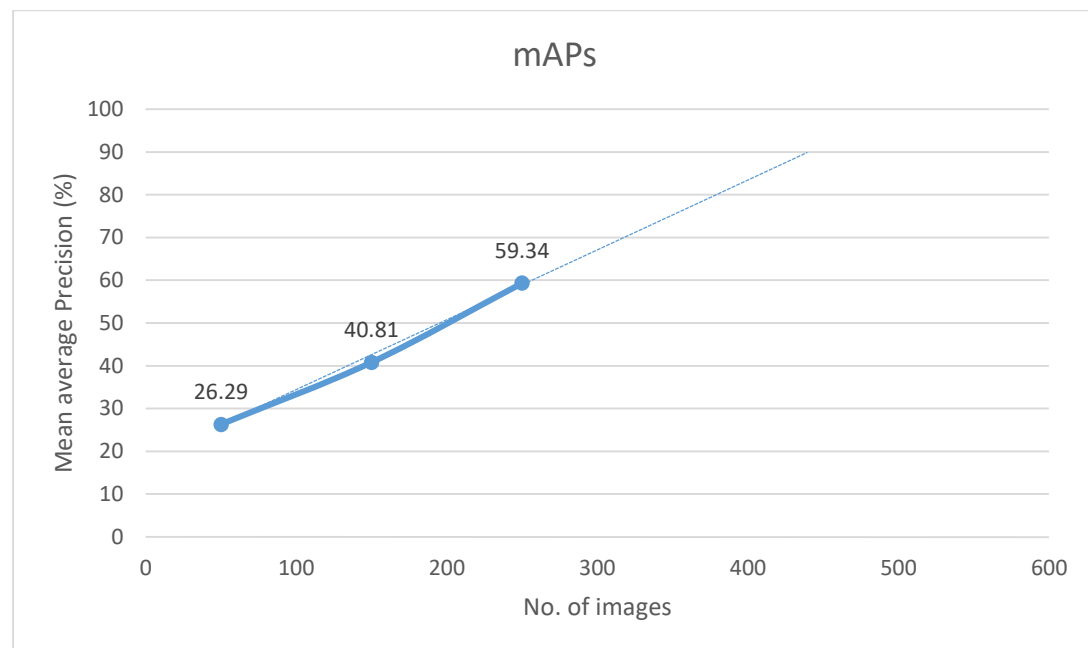


Figure 4.6: Number of Images Used versus Mean Average Precision (%)

From the figure 4.6 above, we can see that if more images are used for training, the higher the accuracy of model can achieve. But when it reaches till certain accuracy level, images would not help much in optimizing anymore. We need to find other ways in order to improve its accuracy. The accuracy not only depend on one factor only, there

are many other factor such as image resolution, data augmentation, architecture of the networks and number of proposal for prediction.

## CHAPTER 5

### CONCLUSION

#### 5.1 Conclusion

The advance of technology which is keep improving over the last decade make the deep learning approach to become possible. Deep learning requires a lots of data to achieve favourable output results and it need a very high specification GPU to reduce the training time. The previous method of machine learning might need lots of hand featured data and must be heavily engineered in order to get high accuracy of object detector. Deep learning requires only images to learn from the image without specifying any feature data. The convolutional network will automatically extract feature and learn from it during the back propagation process. This experiment used Faster RCNN Inception-v2 network to train for the component detection model. It consists a mean average precision of 59.34% which still acceptable. To improve the mAPs, more image data with variety of backgrounds should be collected and input it for the training process. Faster RCNN not suitable for real-time video detection as it detection speed is only around 2FPS. If you looking for real-time video detection, SSD and YOLO might be a good choice as they can run up to 50FPS since their way of detecting object is different. SSD and YOLO does not require any region proposals before they can classify the object. Both of them using multiple network for classifications and localization which help to improve the speed of detecting the objects. Faster RCNN is not only applicable in the industrial area. It also applicable in other field such as medical and biotechnology uses. Those field require this object detection model to help them because of the inspection task that they currently do it manually for micro-organisms is quite challenging. The micro-organisms have different cell shape, color and density. Hence,

it is hard to be identify. Training Faster RCNN model for the use in such field might reduce lots of inspection time and can save many of our human life if certain disease is identified earlier through this detection.

## 5.2 Future Implementation and Recommendation

By using the same technique as this experiment, we could also use it to detect for defectivity of a component. Some of the industrial company are still using manual visual inspection rather than an automated inspection machine. To train the automated inspection machine, we require lots of 'Pass' and 'Fail' component images to do this training in order to have a nice accuracy model. The 'Pass' component images are easier to collect compare to the 'Fail' component images. A component would only consider as defect whenever it is not functionable or not safe to be use. A cracked component is surely will be classify as defect component. In some other cases such as components melt during heating in oven might also classify as defect. The most important uses of the automated inspection machine are help to check the part which require the use microscope. For example, inspection of checking whether the wire is bonded or not. This might hard to see through our eye without a microscope. But with microscope, human sometime might also do some careless mistake and inputting the defect components as usable components. With the help automated inspection machine this could help a lot in the industrial by reducing the need of human workers and reduce some unwanted mistake which done by human. A defect component can be classified to many different kinds. Hence, it is quite challenging in collecting the data for the defect components. In different accidental situation, different kinds of damages could be done on the components. We cannot ensure that all different kinds of accidental damages which done on the components could be collected completely and use it for training purpose. But, we have another deep learning method which known as deep one class method which recognize the 'Pass' component only. It only recognizes those component which is valid to be use. For those components which are different from the normal component product would be classify as abnormal. As the name deep one class method means it only classify and recognize for one class and the other would fall at the category of abnormal. With those abnormal product, we would consider it as a



defect product. Hence, now the only thing is we need to provide those products which consider as valid for the training. Deep learning is the current trend in order to optimize the accuracy of classification as it can react according to its environment and keep learning.

## REFERENCES

- Agarap, A. F. (2019). Deep Learning using Rectified Linear Units (ReLU). [online] Available at: <https://arxiv.org/pdf/1803.08375.pdf> [Accessed 5 Apr. 2019].
- Angelini, C. (2019). GPU Performance Hierarchy 2019: Video Cards Ranked. [online] Available at: <https://www.tomshardware.com/reviews/gpu-hierarchy,4388.html>. [Accessed 5 Apr. 2019].
- Britz, D. (2015). *Understanding Convolutional Neural Networks for NLP*. [online] WildML. Available at: <http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/> [Accessed 10 Apr. 2019].
- Brunette, E. S., Flemmer, R. C. and Flemmer, C. L. (2009). A review of artificial intelligence. ICARA 2009 - Proceedings of the 4th International Conference on Autonomous Robots and Agents, pp. 385–392.
- Clevert, D., Unterthiner, T. and Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (ELUs). [online] Available at: <https://arxiv.org/pdf/1511.07289.pdf> [Accessed 5 Apr. 2019].
- Dayan, P. (2009). Unsupervised learning. The MIT Encyclopedia of the Cognitive Sciences, pp. 1–7.
- Farhadi, F. (2017). Learning Activation Functions in Deep Neural Networks. [online] Available at: [https://publications.polymtl.ca/2945/1/2017\\_FarnoushFarhadi.pdf](https://publications.polymtl.ca/2945/1/2017_FarnoushFarhadi.pdf) [Accessed 5 Apr. 2019].

- He, K., Zhang, X., Ren, S. and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. Proceedings of the IEEE International Conference on Computer Vision. 1026–1034. [online] Available at <https://arxiv.org/pdf/1502.01852.pdf> [Accessed 5 Apr. 2019].
- Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y. Guadarrama, S. and Murphy, K. (2017). Speed/accuracy trade-offs for modern convolutional object detectors. [online] Available at: <https://arxiv.org/pdf/1611.10012.pdf> [Accessed 5 Apr. 2019].
- Hornik, K., Stinchcombe, M. and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2 (5), 359–366.
- Krizhevsky, A., Sutskever, I. and Geoffrey E., H. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25 (NIPS2012)*, pp. 1–9.
- Mandl, S. (2012). What Language Do You Use to Create Your AI Programs and Why?. *KI - Künstliche Intelligenz*, 26, pp. 99–106.
- Mei, X. et al. (2012). Automated optical inspection for die prep. *ASMC (Advanced Semiconductor Manufacturing Conference) Proceedings*, pp. 72–76.
- May, (2017). Deep Learning and the Artificial Intelligence Revolution. *Mongo DB*. [online] Available at: <https://www.mongodb.com/collateral/deep-learning-and-the-artificial-intelligence-revolution> [Accessed 5 Apr. 2019].
- Murphy, J. (2016). An Overview of Convolutional Neural Network Architectures for Deep Learning. *Microway, Inc.*, pp. 1–22. [online] Available at: <https://pdfs.semanticscholar.org/64db/333bb1b830f937b47d786921af4a6c2b3233.pdf> [Accessed 5 Apr. 2019].
- Ongsulee, P. (2017). Artificial Intelligence, Machine Learning and Deep Learning. *Fifteenth International Conference on ICT and Knowledge Engineering*, pp. 1–6.

- O'Shea, K. and Nash, R. (2015) An Introduction to Convolutional Neural Networks. [online] Available at: <https://arxiv.org/pdf/1511.08458.pdf> [Accessed 5 Apr. 2019].
- Padilla, r. (2019). Object-Detection-Metrics. [online] Available at: <https://github.com/rafaelpadilla/Object-Detection-Metrics#create-your-detection-files> [Accessed 5 Apr. 2019].
- Patrick, H. et al. (1998) 'Lecun-98'. IEEE.
- Phon-Amnuaisuk, S., Murata, K.T., Pavarangkoon, P., Yamamoto, K. and Mizuhara, T. (2018). Exploring the Applications of Faster R-CNN and Single-Shot Multi-Box Detection in Smart Nursery Domain. Available at: <https://arxiv.org/pdf/1808.08675.pdf> [Accessed 5 Apr. 2019].
- Raj, B. (2018). A Simple Guide to the Versions of the Inception Network. [online] Available at: <https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202> [Accessed 5 Apr. 2019].
- Shagunsodhani. (2019). Notes for "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" paper. [online] Available at: <https://gist.github.com/shagunsodhani/4441216a298df0fe6ab0> [Accessed 5 Apr. 2019].
- Stutz, D. (2014). Understanding Convolutional Neural Networks. Nips 2016, (3), pp. 1–23.
- Qian, S., Liu, H., Liu, C., Wu, S. and Wong, H. (2018). Adaptive activation functions in convolutional neural networks. *Neurocomputing*, 272, 204–212.
- Ruder, S. (2016). An overview of gradient descent optimization algorithms. pp. 1–14.

- Weng, L. (2017). Object Detection for Dummies Part 3: R-CNN Family. Available at: <https://lilianweng.github.io/lil-log/2017/12/31/object-recognition-for-dummies-part-3.html> [Accessed 5 Apr. 2019].
- Xu, Y. (2018). Faster R-CNN (object detection) implemented by Keras for custom data from Google's Open Images Dataset V4. [online] Towards Data Science. Available at: <https://towardsdatascience.com/faster-r-cnn-object-detection-implemented-by-keras-for-custom-data-from-googles-open-images-125f62b9141a> [Accessed 8 Apr. 2019].
- Ye, R., Pan, C., Chang, M. and Yu, Q. (2018). Intelligent defect classification system based on deep learning. *Advances in Mechanical Engineering*, 10(3), p. 168781401876668.
- Yu, W., Yang, K., Bai, Y., Xiao, T., Yao, H. and Rui, Y. (2016). Visualizing and Comparing AlexNet and VGG using Deconvolutional Layers. *Icml*, 48, pp. 1–7. [online] Available at: <http://www.robots.ox.ac.uk/%0Ahttps://icmlviz.github.io/icmlviz2016/assets/papers/4.pdf> [Accessed 5 Apr. 2019].
- Zeiler, M. D. and Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. *Computer Vision–ECCV 2014*, 8689, pp. 818–833. [online] Available at: <https://arxiv.org/pdf/1311.2901.pdf> [Accessed 5 Apr. 2019].
- Zhang, Chao and Woodland, Philip C (2015). Parameterised sigmoid and relu hidden activation functions for dnn acoustic modelling. *Sixteenth Annual Conference of the International Speech Communication Association*.
- Zhao, Z., Zheng, P., Xu, S. and Wu, X. (2018). Object Detection with Deep Learning: A Review. *Journal of Latex Class Files* 14(8), pp. 1-6. [online] Available at: <https://arxiv.org/pdf/1807.05511.pdf> [Accessed 5 Apr. 2019].

## APPENDICES

### APPENDIX A – Coding

#### Faster RCNN with Inception-v2 (Network Setting)

```

# Faster R-CNN with Inception v2
# Users should configure the fine_tune_checkpoint field in the
train config as
# well as the label_map_path and input_path fields in the
train_input_reader and
# eval_input_reader. Search for "PATH_TO_BE_CONFIGURED" to find
the fields that
# should be configured.

model {
  faster_rcnn {
    num_classes: 5
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 600
        max_dimension: 1024
      }
    }
    feature_extractor {
      type: 'faster_rcnn_inception_v2'
      first_stage_features_stride: 16
    }
    first_stage_anchor_generator {
      grid_anchor_generator {
        scales: [0.25, 0.5, 1.0, 2.0]
        aspect_ratios: [0.5, 1.0, 2.0]
        height_stride: 16
        width_stride: 16
      }
    }
    first_stage_box_predictor_conv_hyperparams {
      op: CONV
      regulariser {
        l2_regularizer {
          weight: 0.0
        }
      }
      initialiser {
        truncated_normal_initializer {
          stddev: 0.01
        }
      }
    }
  }
}

```

```

first_stage_nms_iou_threshold: 0.7
first_stage_max_proposals: 300
first_stage_localisation_loss_weight: 2.0
first_stage_objectness_loss_weight: 1.0
initial_crop_size: 14
maxpool_kernel_size: 2
maxpool_stride: 2
second_stage_box_predictor {
  mask_rcnn_box_predictor {
    use_dropout: false
    dropout_keep_probability: 1.0
    fc_hyperparams {
      op: FC
      regulariser {
        l2_regularizer {
          weight: 0.0
        }
      }
      initialiser {
        variance_scaling_initializer {
          factor: 1.0
          uniform: true
          mode: FAN_AVG
        }
      }
    }
  }
}
}
second_stage_post_processing {
  batch_non_max_suppression {
    score_threshold: 0.0
    iou_threshold: 0.6
    max_detections_per_class: 100
    max_total_detections: 300
  }
  score_converter: SOFTMAX
}
second_stage_localisation_loss_weight: 2.0
second_stage_classification_loss_weight: 1.0
}

train_config: {
  batch_size: 1
  optimiser {
    momentum_optimiser: {
      learning_rate: {
        manual_step_learning_rate {
          initial_learning_rate: 0.0002
          schedule {
            step: 900000
            learning_rate: .00002
          }
          schedule {
            step: 1200000
            learning_rate: .000002
          }
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  gradient_clipping_by_norm: 10.0
  fine_tune_checkpoint:
"C:/Users/User/Desktop/FYP/Tensor1/faster_rcnn_inception_v2_coco_
2016_01_28/model.ckpt"
  from_detection_checkpoint: true
  # Note: The below line limits the training process to 200K
steps, which we
  # empirically found to be sufficient enough to train the pets
dataset. This
  # effectively bypasses the learning rate schedule (the learning
rate will
  # never decay). Remove the below line to train indefinitely.
  num_steps: 200000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
}
}

```

```

train_input_reader: {
  tf_record_input_reader {
    input_path: "C:/Users/User/Desktop/FYP/Tensor1/train.record"
  }
  label_map_path:
"C:/Users/User/Desktop/FYP/Tensor1/training/labelmap.pbtxt"
}

eval_config: {
  num_examples: 54
  # Note: The below line limits the evaluation process to 10
evaluations.
  # Remove the below line to evaluate indefinitely.
  max_evals: 10
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "C:/Users/User/Desktop/FYP/Tensor1/test.record"
  }
  label_map_path:
"C:/Users/User/Desktop/FYP/Tensor1/training/labelmap.pbtxt"
  shuffle: false
  num_readers: 1
}

```

## PASCAL VOC 2012 (Calculate Mean Average Precision)

```

import argparse
import glob
import os
import shutil
# from argparse import RawTextHelpFormatter
import sys

import _init_paths
from BoundingBox import BoundingBox
from BoundingBoxes import BoundingBoxes
from Evaluator import *
from utils import BBFormat

# Validate formats
def ValidateFormats(argFormat, argName, errors):
    if argFormat == 'xywh':
        return BBFormat.XYWH
    elif argFormat == 'xyrb':
        return BBFormat.XYX2Y2
    elif argFormat is None:
        return BBFormat.XYWH # default when nothing is passed
    else:
        errors.append(
            'argument %s: invalid value. It must be either \'xywh
\' or \'xyrb\'' % argName)

# Validate mandatory args
def ValidateMandatoryArgs(arg, argName, errors):
    if arg is None:
        errors.append('argument %s: required argument' % argName)
    else:
        return True

```



```

def ValidateImageSize(arg, argName, argInformed, errors):
    errorMsg = 'argument %s: required argument if %s is relative'
    # (argName, argInformed)
    ret = None
    if arg is None:
        errors.append(errorMsg)
    else:
        arg = arg.replace('(', '').replace(')', '')
        args = arg.split(',')
        if len(args) != 2:
            errors.append(
                '%s. It must be in the format \'width,height\'
(e.g. \'600,400\')' % errorMsg)
        else:
            if not args[0].isdigit() or not args[1].isdigit():
                errors.append(
                    '%s. It must be in INdiaTEGER the format
\'width,height\' (e.g. \'600,400\')' %
                    errorMsg)
            else:
                ret = (int(args[0]), int(args[1]))
    return ret

# Validate coordinate types
def ValidateCoordinatesTypes(arg, argName, errors):
    if arg == 'abs':
        return CoordinatesType.Absolute
    elif arg == 'rel':
        return CoordinatesType.Relative
    elif arg is None:
        return CoordinatesType.Absolute # default when nothing
is passed
    errors.append('argument %s: invalid value. It must be either
\'rel\' or \'abs\'' % argName)

def ValidatePaths(arg, nameArg, errors):
    if arg is None:
        errors.append('argument %s: invalid directory' % nameArg)
    elif os.path.isdir(arg) is False and
os.path.isdir(os.path.join(currentPath, arg)) is False:
        errors.append('argument %s: directory does not exist \'%s
\'' % (nameArg, arg))
    # elif os.path.isdir(os.path.join(currentPath, arg)) is True:
    #     arg = os.path.join(currentPath, arg)
    else:
        arg = os.path.join(currentPath, arg)
    return arg

```

```

def getBoundingBoxes(directory,
                    isGT,
                    bbFormat,
                    coordType,
                    allBoundingBoxes=None,
                    allClasses=None,
                    imgSize=(0, 0)):
    """Read txt files containing bounding boxes (ground truth and
    detections)."""
    if allBoundingBoxes is None:
        allBoundingBoxes = BoundingBoxes()
    if allClasses is None:
        allClasses = []
    # Read ground truths
    os.chdir(directory)
    files = glob.glob("*.txt")
    files.sort()
    # Read GT detections from txt file
    # Each line of the files in the groundtruths folder
    # represents a ground truth bounding box
    # (bounding boxes that a detector should detect)
    # Each value of each line is "class_id, x, y, width, height"
    # respectively
    # Class_id represents the class of the bounding box
    # x, y represents the most top-left coordinates of the
    # bounding box
    # x2, y2 represents the most bottom-right coordinates of the
    # bounding box
    for f in files:
        nameOfImage = f.replace(".txt", "")
        fh1 = open(f, "r")
        for line in fh1:
            line = line.replace("\n", "")
            if line.replace(' ', '') == '':
                continue
            splitLine = line.split(" ")

            if isGT:
                # idClass = int(splitLine[0]) #class
                idClass = (splitLine[0]) # class
                x = float(splitLine[1])
                y = float(splitLine[2])
                w = float(splitLine[3])
                h = float(splitLine[4])
                bb = BoundingBox(
                    nameOfImage,
                    idClass,
                    x,
                    y,
                    w,
                    h,
                    coordType,
                    imgSize,
                    BBType.GroundTruth,
                    format=bbFormat)
            else:
                # idClass = int(splitLine[0]) #class
                idClass = (splitLine[0]) # class
                confidence = float(splitLine[1])
                x = float(splitLine[2])
                y = float(splitLine[3])
                w = float(splitLine[4])
                h = float(splitLine[5])
                bb = BoundingBox(
                    nameOfImage,
                    idClass,
                    x,
                    y,
                    w,
                    h,
                    coordType,
                    imgSize,
                    BBType.Detected,
                    confidence,
                    format=bbFormat)

```

```

        allBoundingBoxes.addBoundingBox(bb)
        if idClass not in allClasses:
            allClasses.append(idClass)
    fh1.close()
    return allBoundingBoxes, allClasses

# Get current path to set default folders
currentPath = os.path.dirname(os.path.abspath(__file__))

VERSION = '0.1 (beta)'

parser = argparse.ArgumentParser(
    prog='Object Detection Metrics - Pascal VOC',
    description='This project applies the most popular metrics
used to evaluate object detection '
    'algorithms.\n\nThe current implementation runs the Pascal VOC
metrics.\n\nFor further references, '
    'please check:\n\nhttps://github.com/rafaelpadilla/Object-
Detection-Metrics',
    epilog="Developed by: Rafael Padilla
(rafael.padilla@smt.ufrj.br)")
# formatter_class=RawTextHelpFormatter)
parser.add_argument('-v', '--version', action='version',
version='% (prog)s ' + VERSION)
# Positional arguments
# Mandatory
parser.add_argument(
    '-gt',
    '--gtfolder',
    dest='gtFolder',
    default=os.path.join(currentPath, 'groundtruths'),
    metavar='',
    help='folder containing your ground truth bounding boxes')
parser.add_argument(
    '-det',
    '--detfolder',
    dest='detFolder',
    default=os.path.join(currentPath, 'detections'),
    metavar='',
    help='folder containing your detected bounding boxes')

# Optional
parser.add_argument(
    '-t',
    '--threshold',
    dest='iouThreshold',
    type=float,
    default=0.5,
    metavar='',
    help='IOU threshold. Default 0.5')
parser.add_argument(
    '-gtformat',
    dest='gtFormat',
    metavar='',
    default='xywh',
    help='format of the coordinates of the ground truth bounding
boxes: '
    '\(\backslash'xywh\': <left> <top> <width> <height>)'
    ' or (\backslash'xyrb\': <left> <top> <right> <bottom>)' )
parser.add_argument(
    '-detformat',
    dest='detFormat',
    metavar='',
    default='xywh',
    help='format of the coordinates of the detected bounding
boxes: '
    '\(\backslash'xywh\': <left> <top> <width> <height>)'
    ' or (\backslash'xyrb\': <left> <top> <right> <bottom>)' )
parser.add_argument(
    '-gtcoords',
    dest='gtCoordinates',
    default='abs',
    metavar='',
    help='reference of the ground truth bounding box coordinates:
absolute '
    'values (\backslash'abs\') or relative to its image size (\backslash'rel\')')
parser.add_argument(
    '-detcoords',
    default='abs',
    dest='detCoordinates',
    metavar='',
    help='reference of the ground truth bounding box coordinates:

```

```

    'absolute values (\'abs\)\' or relative to its image size
    (\'rel\')')
parser.add_argument(
    '-imgsize',
    dest='imgSize',
    metavar='',
    help='image size. Required if -gtcoords or -detcoords are
    \'rel\')')
parser.add_argument(
    '-sp', '--savepath', dest='savePath', metavar='',
    help='folder where the plots are saved')
parser.add_argument(
    '-np',
    '--noplot',
    dest='showPlot',
    action='store_false',
    help='no plot is shown during execution')
args = parser.parse_args()

iouThreshold = args.iouThreshold

# Arguments validation
errors = []
# Validate formats
gtFormat = ValidateFormats(args.gtFormat, '-gtformat', errors)
detFormat = ValidateFormats(args.detFormat, '-detformat', errors)
# Groundtruth folder
if ValidateMandatoryArgs(args.gtFolder, '-gt/--gtfolder',
errors):
    gtFolder = ValidatePaths(args.gtFolder, '-gt/--gtfolder',
errors)
else:
    # errors.pop()
    gtFolder = os.path.join(currentPath, 'groundtruths')
    if os.path.isdir(gtFolder) is False:
        errors.append('folder %s not found' % gtFolder)
# Coordinates types
gtCoordType = ValidateCoordinatesTypes(args.gtCoordinates, '-
gtCoordinates', errors)
detCoordType = ValidateCoordinatesTypes(args.detCoordinates, '-
detCoordinates', errors)

imgSize = (0, 0)
if gtCoordType == CoordinatesType.Relative: # Image size is
required
    imgSize = ValidateImageSize(args.imgSize, '-imgsize', '-
gtCoordinates', errors)
if detCoordType == CoordinatesType.Relative: # Image size is
required
    imgSize = ValidateImageSize(args.imgSize, '-imgsize', '-
detCoordinates', errors)
# Detection folder
if ValidateMandatoryArgs(args.detFolder, '-det/--detfolder',
errors):
    detFolder = ValidatePaths(args.detFolder, '-det/--detfolder',
errors)
else:
    # errors.pop()
    detFolder = os.path.join(currentPath, 'detections')
    if os.path.isdir(detFolder) is False:
        errors.append('folder %s not found' % detFolder)
if args.savePath is not None:
    savePath = ValidatePaths(args.savePath, '-sp/--savepath',
errors)
else:
    savePath = os.path.join(currentPath, 'results')
# Validate savePath
# If error, show error messages
if len(errors) is not 0:
    print("""usage: Object Detection Metrics [-h] [-v] [-gt] [-
det] [-t] [-gtformat]
                                [-detformat] [-save]""")
    print('Object Detection Metrics: error(s): ')
    [print(e) for e in errors]
    sys.exit()

# Create directory to save results
shutil.rmtree(savePath, ignore_errors=True) # Clear folder
os.makedirs(savePath)
# Show plot during execution
showPlot = args.showPlot

```

```

# print('iouThreshold= %f' % iouThreshold)
# print('savePath = %s' % savePath)
# print('gtFormat = %s' % gtFormat)
# print('detFormat = %s' % detFormat)
# print('gtFolder = %s' % gtFolder)
# print('detFolder = %s' % detFolder)
# print('gtCoordType = %s' % gtCoordType)
# print('detCoordType = %s' % detCoordType)
# print('showPlot %s' % showPlot)

# Get groundtruth boxes
allBoundingBoxes, allClasses = getBoundingBoxes(
    gtFolder, True, gtFormat, gtCoordType, imgSize=imgSize)
# Get detected boxes
allBoundingBoxes, allClasses = getBoundingBoxes(
    detFolder, False, detFormat, detCoordType, allBoundingBoxes,
    allClasses, imgSize=imgSize)
allClasses.sort()

evaluator = Evaluator()
acc_AP = 0
validClasses = 0

# Plot Precision x Recall curve
detections = evaluator.PlotPrecisionRecallCurve(
    allBoundingBoxes, # Object containing all bounding boxes
    (ground truths and detections)
    IOUThreshold=iouThreshold, # IOU threshold
    method=MethodAveragePrecision.EveryPointInterpolation,
    showAP=True, # Show Average Precision in the title of the
    plot
    showInterpolatedPrecision=False, # Don't plot the
    interpolated precision curve
    savePath=savePath,
    showGraphic=showPlot)

f = open(os.path.join(savePath, 'results.txt'), 'w')
f.write('Object Detection Metrics\n')
f.write('https://github.com/rafaelpadilla/Object-Detection-
Metrics\n\n')
f.write('Average Precision (AP), Precision and Recall per
class:')

# each detection is a class
for metricsPerClass in detections:

    # Get metric values per each class
    cl = metricsPerClass['class']
    ap = metricsPerClass['AP']
    precision = metricsPerClass['precision']
    recall = metricsPerClass['recall']
    totalPositives = metricsPerClass['total positives']
    total_TP = metricsPerClass['total TP']
    total_FP = metricsPerClass['total FP']

    if totalPositives > 0:
        validClasses = validClasses + 1
        acc_AP = acc_AP + ap
        prec = ['%.2f' % p for p in precision]
        rec = ['%.2f' % r for r in recall]
        ap_str = "{0:.2f}%".format(ap * 100)
        # ap_str = "{0:.4f}%".format(ap * 100)
        print('AP: %s (%s)' % (ap_str, cl))
        f.write('\n\nClass: %s' % cl)
        f.write('\n\nAP: %s' % ap_str)
        f.write('\n\nPrecision: %s' % prec)
        f.write('\n\nRecall: %s' % rec)

mAP = acc_AP / validClasses
mAP_str = "{0:.2f}%".format(mAP * 100)
print('mAP: %s' % mAP_str)
f.write('\n\nmAP: %s' % mAP_str)

```

---

## Training Session

```
# Copyright 2017 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the
# License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
# software
# distributed under the License is distributed on an "AS IS"
# BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions
# and
# limitations under the License.
#
=====
=====

r"""Training executable for detection models.

This executable is used to train DetectionModels. There are two
ways of
configuring the training job:

1) A single pipeline_pb2.TrainEvalPipelineConfig configuration
file
can be specified by --pipeline_config_path.

Example usage:
./train \
  --logtostderr \
  --train_dir=path/to/train_dir \
  --pipeline_config_path=pipeline_config.pbtxt
```

2) Three configuration files can be provided: a `model_pb2.DetectionModel` configuration file to define what type of `DetectionModel` is being trained, an `input_reader_pb2.InputReader` file to specify what training data will be used and a `train_pb2.TrainConfig` file to configure training parameters.

Example usage:

```
./train \
  --logtostderr \
  --train_dir=path/to/train_dir \
  --model_config_path=model_config.pbtxt \
  --train_config_path=train_config.pbtxt \
  --input_config_path=train_input_config.pbtxt
'''

import functools
import json
import os
import tensorflow as tf

from object_detection.builders import dataset_builder
from object_detection.builders import graph_rewriter_builder
from object_detection.builders import model_builder
from object_detection.legacy import trainer
from object_detection.utils import config_util

tf.logging.set_verbosity(tf.logging.INFO)

flags = tf.app.flags
flags.DEFINE_string('master', '', 'Name of the TensorFlow master
to use.')
flags.DEFINE_integer('task', 0, 'task id')
flags.DEFINE_integer('num_clones', 1, 'Number of clones to deploy
per worker.')
flags.DEFINE_boolean('clone_on_cpu', False,
                    'Force clones to be deployed on CPU. Note
that even if '
                    'set to False (allowing ops to run on gpu),
some ops may '
                    'still be run on the CPU if they have no GPU
kernel.')
flags.DEFINE_integer('worker_replicas', 1, 'Number of
worker+trainer '
                    'replicas.')
flags.DEFINE_integer('ps_tasks', 0,
                    'Number of parameter server tasks. If None,
does not use '
                    'a parameter server.')
flags.DEFINE_string('train_dir', '',
                    'Directory to save the checkpoints and
training summaries.')
flags.DEFINE_string('pipeline_config_path', '',
                    'Path to a
pipeline_pb2.TrainEvalPipelineConfig config '
                    'file. If provided, other configs are
ignored')
flags.DEFINE_string('train_config_path', '',
                    'Path to a train_pb2.TrainConfig config
file.')
flags.DEFINE_string('input_config_path', '',
                    'Path to an input_reader_pb2.InputReader
config file.')
flags.DEFINE_string('model_config_path', '',
                    'Path to a model_pb2.DetectionModel config
file.')

FLAGS = flags.FLAGS
```





```

    worker_job_name = '%s/task:%d' % (task_info.type,
task_info.index)
    task = task_info.index
    is_chief = (task_info.type == 'master')
    master = server.target

graph_rewriter_fn = None
if 'graph_rewriter_config' in configs:
    graph_rewriter_fn = graph_rewriter_builder.build(
        configs['graph_rewriter_config'], is_training=True)

trainer.train(
    create_input_dict_fn,
    model_fn,
    train_config,
    master,
    task,
    FLAGS.num_clones,
    worker_replicas,
    FLAGS.clone_on_cpu,
    ps_tasks,
    worker_job_name,
    is_chief,
    FLAGS.train_dir,
    graph_hook_fn=graph_rewriter_fn)

if __name__ == '__main__':
    tf.app.run()

```

---

## Testing Session (With GUI)

```

import tkinter
import cv2
import os
import PIL.Image
import PIL.ImageTk
import time
import numpy as np
import tensorflow as tf
import sys

# Import utilites
from utils import label_map_util
from utils import visualization_utils as vis_util

import warnings
warnings.filterwarnings('ignore')

Realtime = 'C:\\Users\\USER\\Desktop\\FYP\\Tensor1'
os.chdir(Realtime)

class MyVideoCapture:
    def __init__(self, video_source=1):
        self.vid = cv2.VideoCapture(video_source)
        if not self.vid.isOpened():
            raise ValueError("Unable to open video source",
video_source)

        self.width = self.vid.get(cv2.CAP_PROP_FRAME_WIDTH)
        self.height = self.vid.get(cv2.CAP_PROP_FRAME_HEIGHT)

```

```

def get_frame(self):
    if self.vid.isOpened():
        ret, frame = self.vid.read()
        if ret:
            frame = cv2.resize(frame, (640,480))
            return (ret, cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB))
        else:
            return (ret, None)
    else:
        return (ret, None)

def __del__(self):
    if self.vid.isOpened():
        self.vid.release()
    self.window.mainloop()

##-----
class App:
    def __init__(self, window, window_title, video_source=1):

        self.window = window
        self.window.title(window_title)
        self.video_source = video_source

        self.vid = MyVideoCapture(video_source)

        self.frame1 = tkinter.LabelFrame(window, text="Real Time
Image", width=640, height=480, bd=5)
        self.frame2 = tkinter.LabelFrame(window, text="Result",
width=640, height=480, bd=5)

        self.frame1.grid(row=0, column=0, columnspan=2,
sticky="W", padx=10, pady=10)
        self.frame2.grid(row=0, column=1, columnspan=2,
sticky="W", padx=680)

        self.canvas = tkinter.Canvas(self.frame1, width = 640,
height = 480)
        self.canvas.grid()

        self.canvas2 = tkinter.Canvas(self.frame2, width = 640,
height = 480)
        self.canvas2.grid()

        self.btn_test=tkinter.Button(window, text="Test", width=
50, command=self.test)
        self.btn_test.grid(row=1, column=1, columnspan=2,
sticky="W", padx=150, pady=10)
        self.btn_test.config(font=("Baskerville Old Face",25))

        self.delay = 15
        self.update()

        self.window.mainloop()

def update(self):
    ret, frame = self.vid.get_frame()

```

```

        if ret:
            self.photo = PIL.ImageTk.PhotoImage(image =
PIL.Image.fromarray(frame))
            self.canvas.create_image(0, 0, image =
self.photo, anchor="nw")

        self.window.after(self.delay, self.update)

    def test(self):

        ret, frame = self.vid.get_frame()
        if ret:
            cv2.imwrite("test1.jpg", cv2.cvtColor(frame,
cv2.COLOR_RGB2BGR))

            image_files = os.listdir(Realtime)
            latest = max(image_files, key=os.path.getctime)
            test_image = cv2.imread(latest)
            sys.path.append("../")
            MODEL_NAME='inference_graph'
            IMAGE_NAME='test1.jpg'
            CWD_PATH= os.getcwd()

PATH_TO_CKPT=os.path.join(CWD_PATH,MODEL_NAME,'frozen_inference_g
raph.pb')

PATH_TO_LABELS=os.path.join(CWD_PATH,'training','labelmap.pbtxt')
    PATH_TO_IMAGE=os.path.join(CWD_PATH,IMAGE_NAME)
    NUM_CLASSES=5
    label_map=label_map_util.load_labelmap(PATH_TO_LABELS)
    categories=
label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES, use_display_name=True)

category_index=label_map_util.create_category_index(categories)
    detection_graph=tf.Graph()
    with detection_graph.as_default():
        od_graph_def=tf.GraphDef()
        with tf.gfile.GFile(PATH_TO_CKPT,'rb') as fid:
            serialized_graph=fid.read()
            od_graph_def.ParseFromString(serialized_graph)
            tf.import_graph_def(od_graph_def,name='')
        sess=tf.Session(graph=detection_graph)

image_tensor=detection_graph.get_tensor_by_name('image_tensor:0')

detection_boxes=detection_graph.get_tensor_by_name('detection_box
es:0')

detection_scores=detection_graph.get_tensor_by_name('detection_sc
ores:0')

detection_classes=detection_graph.get_tensor_by_name('detection_c
lasses:0')

num_detections=detection_graph.get_tensor_by_name('num_detections
:0')

```

```

        image=cv2.imread(PATH_TO_IMAGE)
        image_expanded=np.expand_dims(image,axis=0)
        (boxes, scores, classes, num) =
sess.run([detection_boxes,detection_scores,detection_classes,num_
detections],feed_dict={image_tensor:image_expanded})
        vis_util.visualize_boxes_and_labels_on_image_array(
            image,
            np.squeeze(boxes),
            np.squeeze(classes).astype(np.int32),
            np.squeeze(scores),
            category_index,
            use_normalized_coordinates=True,
            line_thickness=8,
            min_score_thresh=0.80)
        cv2.imwrite("frame-" + time.strftime("%d-%m-%Y-%H-%
M-%S") + ".jpg", cv2.cvtColor(image, cv2.COLOR_RGB2BGR))
        processed_files = os.listdir(Realtime)
        latest_processed = max(processed_files,
key=os.path.getctime)

        results = cv2.imread(latest_processed)

        self.photo2=PIL.ImageTk.PhotoImage(image=
PIL.Image.fromarray(results))
        self.canvas2.create_image(0, 0,
image=self.photo2,anchor="nw")
App(tkinter.Tk(), "Components Identifier")

```