AUTOMATED VISION INSPECTION BASED IC COMPONENT LOCATOR USING DEEP LEARNING

IBRAHIM SOLIMAN MOHAMED

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

AUTOMATED VISION INSPECTION BASED IC COMPONENT LOCATOR USING DEEP LEARNING

IBRAHIM SOLIMAN MOHAMED

This report is submitted in partial fulfilment of the requirements for the degree of Bachelor of Electronic Engineering with Honours

Faculty of Electronic and Computer Engineering
Universiti Teknikal Malaysia Melaka

JUNE 2019

UNIVERSITI TEKNIKAL MALAYSIA MELAKA
FAKULTI KEJUTERAAN ELEKTRONIK DAN KEJURUTERAAN KOMPUTER

BORANG PENGESAHAN STATUS LAPORAN
PROJEK SARJANA MUDA II

Tajuk Projek : AUTOMATED VISION INSPECTION BASED IC COMPONENT LOCATOR USING DEEP LEARNING

Sesi Pengajian : 2018/2019

Saya IBRAHIM SOLIMAN MOHAMED mengaku membenarkan laporan Projek Sarjana Muda ini disimpan di Perpustakaan dengan syarat-syarat kegunaan seperti berikut:

1. Laporan adalah hakmilik Universiti Teknikal Malaysia Melaka.
2. Perpustakaan dibenarkan membuat salinan untuk tujuan pengajian sahaja.
3. Perpustakaan dibenarkan membuat salinan laporan ini sebagai bahan pertukaran antara institusi pengajian tinggi.
4. Sila tandakan (✓):

☐ **SULIT*** (Mengandungi maklumat yang berdarjah keselamatan atau kepentingan Malaysia seperti yang termaktub di dalam AKTA RAHSIA RASMI 1972)

☐ **TERHAD*** (Mengandungi maklumat terhad yang telah ditentukan oleh organisasi/badan di mana penyelidikan dijalankan.

☐ **TIDAK TERHAD**

Disahkan oleh:

(TANDATANGAN PENULIS) (COP DAN TANDATANGAN PENYELIA)

Alamat Tetap: .....

Tarikh : ..... Tarikh : .....

*CATATAN: Jika laporan ini SULIT atau TERHAD, sila lampirkan surat daripada pihak berkuasa/organisasi berkenaan dengan menyatakan sekali tempoh laporan ini perlu dikelaskan sebagai SULIT atau TERHAD.

DECLARATION

I declare that this report entitled “AUTOMATED VISION INSPECTION BASED IC COMPONENT LOCATOR USING DEEP LEARNING” is the result of my own work except for quotes as cited in the references.

Signature : ……

Author : ……

Date : ……

APPROVAL

I hereby declare that I have read this thesis and in my opinion this thesis is sufficient in terms of scope and quality for the award of Bachelor of Electronic Engineering with Honours.

Signature	:	……
Supervisor Name	:	PROF. DR. ZULKALNAIN BIN MOHD YUSSOF
Date	:	……

DEDICATION

For my beloved father and mother.

ABSTRACT

With the coming of the era of industrial revolution 4.0, manufacturers produce high-tech products. As the production process is refined, inspection technologies become more important. Specifically, the inspection of printed circuit board (PCB), which is an indispensable part of electronic products, is an essential step to improve the quality of the process and yield. Image processing techniques are typically utilized for inspection, but there are limitations because the backgrounds of images are different, and the components shape and size parameters are normally various. In order to overcome these limitations, methods based on machine and deep learning have been developed recently. In this project, an IC components locator software is developed to help in inspection process, this software relies on the two most popular model of object detection based on deep learning field (Yolo V3 and Faster RCNN). Both models were trained and a performance comparison was made between their results in terms of mAP, loss, inference time and training time. Yolo V3 and Faster RCNN were trained on a filtered open source PCB dataset that contains 163 Images. An annotation and augmentation tool has also been developed for the purpose of increasing the amount of our dataset. Finally, OPENVINO toolkit has been used for optimization process, in which both models were inferred on various Intel CPUs to test their performance.

ACKNOWLEDGEMENTS

First and foremost, I would like to acknowledge and express my utmost gratitude to my supervisor Professor Dr. Zulkalnain Bin Mohd Yussof from Faculty of Electronics and Computer Engineering, Universiti Teknikal Malaysia Melaka (UTeM) for his essential guidance and supervision throughout the research of this final year project. I am grateful for he has generously shared his experience and valuable knowledge, the unconditional support and encouragement towards the completion of this thesis.

Finally, I would like to thank my peers, my beloved family for their moral support in completing this final year project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

2D	:	2 Dimensions
ALU	:	Arithmetic Logic Unit
AOI	:	Automated Optical Inspection
API	:	Application Programming Interface
AVI	:	Automated Vision Inspection
BFLOP	:	Billion Floating Point Operations
CNN	:	Convolutional Neural Network
CPU	:	Central Processing Unit
DLDT	:	Deep Learning Deployment Toolkit
DNN	:	Deep Neural Network
DSLR	:	Digital Single-Lens Reflex
FN	:	False Negative
FP	:	False Positive
FPGA	:	Field-Programmable Gate Array
FPN	:	Feature Pyramid Networks
FVI	:	Final Vision Inspection
GPU	:	Graphical Processing Unit
HOG	:	Histogram of Oriented Gradients
IC	:	Integrated Circuit

IoU	:	Intersection of Union
IPU	:	Imaging Processing Unit
IR	:	Intermediate Representation
JPG	:	Joint Photographic Group
mAP	:	Mean Average Precision
PCB	:	Printed Circuit Board
PNG	:	Portable Network Graphics
R-CNN	:	Region Convolutional Neural Network
ROI	:	Region of Interests
RPN	:	Region Proposal Network
SIFT	:	Scale-Invariant Feature Transform
SSD	:	Single Shot Multibox Detector
SVM	:	Support Vector Machine
TP	:	True Positive
UTeM	:	Universiti Teknikal Malaysia Melaka
VPU	:	Vision Processing Unit
YoloV3	:	You Only Look Once Version 3

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Project Background

The machine-controlled inspection of Printed Circuit Boards (PCBs) serves a purpose to alleviate human inspectors of the tedious and inefficient task of trying to find those defects in PCBs that could lead to electrical failure. As an example, circuit breaks have rather obvious implications for electrical failure, and human inspectors typically miss those defects. It is merely tedious to visually examine many thousands of printed wires with parts placement, every couple of thousandths of a feet across, for several hours each day without any mistakes or misses that can cause a lot of problem in production line. Such mistakes, whereas dead perceivable, also are pricey. With the rapid-movement limitations of microscopic viewing, the unskillfulness of human scrutiny is going to be intolerable. Automated, machine based mostly, scrutiny relieves

this drawback by providing a machine computer vision solution. Obviously, there are managerial, employment, and wide-ranging economic implications of such technology which must be considered along with the technology itself. So, Inspection automation has become a unique and important tool to boost quality in printed circuit board (PCB) manufacturer factories. Nowadays Manufacturing Industries need machine-controlled inspection since, within the fabrication processes, the square measure uncertainties, tolerances, defects, relative position and orientation errors, can be analyzed by vision sensing, Machine algorithms and Deep Neural Networks.

This project proposes an approach for printed circuit board (PCB) component locator using the power of most recent deep learning networks that could achieve an impressive result in automated inspection. the main function of this project is generating data that includes the location, dimensions and categories of each component, which can help in generating inspection board recipe. Currently this recipe is created for each new board in production line manually by operator using a custom-made bounding box drawing mechanism and type selection for each component.

In this project, a comparison between YoloV3 [8] and Faster R-CNN [9] object detection networks will be made as a component locator by training both networks on provided electronics board datasets [10].

1.2 Problem Statement

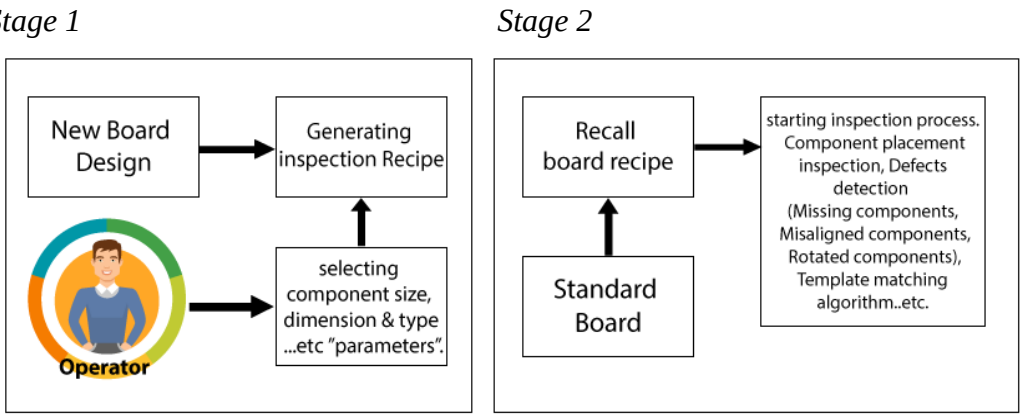


Figure 1.1: Part of PCB Inspection Process

As shown in Figure 1.1, stage 1 shows the procedures that the operator follows for a new board in their production line. The operator needs to draw a bounding box around each component and select component's family, type and other parameters that can help in specifying suitable inspection algorithm. This operation is time costly and needs much efforts and concentration of the operator. So, this project will focus on generating new board recipe or even help in automating part of this inspection stage by allocating, localizing each component and classify some this component. On the other hand, the second stage is nearly automated by a lot of popular algorithms in imaging processing fields, and it is nearly stable and required less operator efforts and time.

1.3 Objectives of the Research

i. To develop annotation and augmentation software for dataset preparation.

ii. To train, test and compare IC component locator based on object detection network (YoloV3 vs Faster R-CNN)

iii. To deploy our IC component locator on Intel CPU computer using OPENVINO software development kit.

1.4 Scope of Work

In purpose of automating the inspection process of PCB on production line, after days of observation on the operator inspection that use a manual technique for each new designed board at production line, operator has been observed select and draw bounding box for each component and selecting the suitable inspection algorithm according to the selected component, so by developing an automated component locator using DNN and CNN, we will be able to save the operator time and effort by automatically identify the ROI for each component, that will help operator in choosing the suitable inspection algorithm in faster way. My network will be developed on Tensorflow or caffe2 framework using python programming language in purpose of reaching ±10 px localization of component body size. Finally, we will optimize our network and deploy it on intel CPU computer using OPENVINO software development kit.

1.5 Report Structure

This thesis is organized and arranged into 5 major chapters. In chapter 1, the overview of PCB inspection is discussed in the project background. In addition, the problem statement, objective and scope of work will be outlined in this section. In chapter 2, the past studies related to PCB automated inspection will be included in this chapter. In chapter 3, all relevant experiments and techniques used in this project will be mentioned in details. The flowchart of the system will be discussed as well. In chapter 4, the performance of our system will be recorded and interpreted in terms of accuracy, computation time and reliability. In the last chapter, a conclusion will be drawn from this project. In addition, the recommendation for the future plan related to the project will be made in this section.

CHAPTER 2

BACKGROUND STUDY

2.1 Inspection Categories

In PCB manufacturing industries, Optical inspection has been widely used in the past few decades. It is currently serving a very important role in fabrication and mass production process. Most PCB manufacturing players are currently relying on AOI machines to detect and report different kind of defects on boards when photo-printing or etching. Notwithstanding, AVI (Automated Vision Inspection) that generally additionally referred to as FVI (final vision inspection) is growing in an exceedingly comparatively quick pace, however not nevertheless widely used in the market. AOI and AVI machines are different categories of inspection machines with different function, however their operating concept still similar. several technologies of AVI are designed from AOI. The PCB makers are currently using AOI and AVI

machines to make sure of standard and precise inspection process of their boards. Both machines are relying on cameras input image to visually detect the defects.

2.1.1 **AOI,** it is a category of machine-controlled optical Inspection PCB throughout the fabrication and production process. its function is scanning the inner layers and outer layers of PCB when the processes of etching and baring. when scanning by AOI machine, the defects that do not meet the manufacturer's demand are going to be detected by the machine. this means, AOI will detect and alert about issues early within the production time, thus faults part would not be passed to next production step and a huge cost may be saved. Therefore, AOI will increase the standard and precision of PCBs manufacturing inspection and scale down the assembly cost over fabrication and production time.

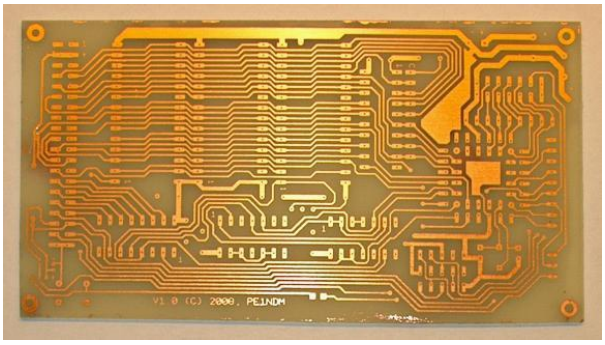


Figure 2.1: Printed Circuit Board after Fabrication

Figure 2.2: AOI Machine

2.1.2 **AVI,** Automated vision inspection is the second categories of inspection that we have studied, it is a machine-controlled vision review on the final PCB product. in contrast to AOI that is an intermediate review stage of PCB. Therefore, AVI is additionally referred to as final vision review. Before shipping out of PCBs, cosmetic review which might detect the defects on the overall final board appearance is extremely necessary. These defects were traditionally checked by human eyes. However, with AVI technology, several makers use AVI machines to automate the final inspection. Compared to human eyes checking, AVI machines will offer quicker and more precise vision inspection. Therefore, nowadays all PCB makers are moving rapidly to use AVI machines to enhance their PCB's production line inspection.

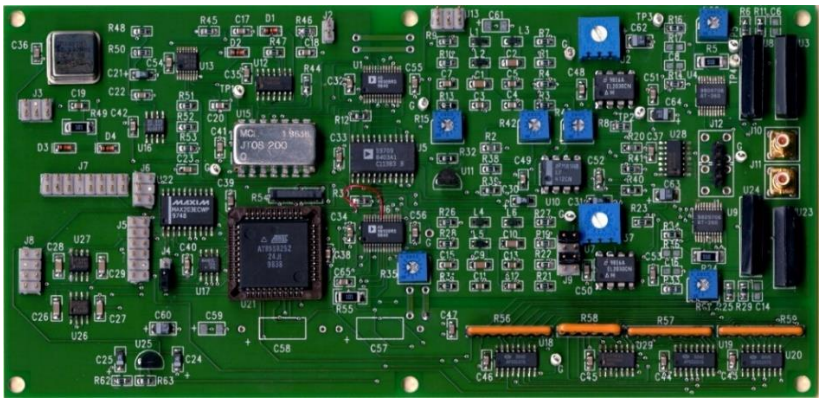


Figure 2.3: PCB Sample for AVI

Figure 2.4: AVI Machine

2.2 Inspection Techniques

PCB Manual Inspection is a vital a part of any electronics production facility, but once the production volume and inspection time by operator increase, the operator will suffer from fatigue which we can simply expect to not see a problem or to even miss complete components of the assembly. Machine-driven inspection is commonplace within the production space. AOI (Automated Optical Inspection) in 2D, 3D or x-ray is employed for selective or 100% inspection of electronics parts. Many various image processing techniques are bestowed to unravel the matter of locating and classifying elements like image subtraction [1], [7] and feature-based approaches [2]. Alternative approaches square measure supported templet matching [6] has been developed. Some approaches that use morphological operations have conjointly been developed too [3,4,5].

2.2.1 Pixel subtraction or Image subtraction operator requires 2 pictures as input and output a third image which its pixels representation is merely those of the primary image (1st image) subtracted from the corresponding image (2nd image). An image subtraction can also be applied using one image as input then performs a subtraction operation using a constant value on all pixel values of this image. Some types of the operator can simply output the difference between each pixel values absolutely, instead of the simple signed output.

The subtraction of 2 pictures is performed in straightforward method in an exceedingly single pass. The output picture pixel values are given by:

$$Q(i,j) = P_1(i,j) - P_2(i,j)$$

2.2.2 **Morphological Image Process** defined as a group or sequence of non-linear operations associated with the form or morphology of unique feature in a picture. consistent with Wikipedia, morphological operations are heavy depending on the relative ordering of pixel values, not on their numerical values, and so area unit particularly suited to the process of binary pictures. Morphological operations are able to be applied to greyscale pictures such their lightweight transfer functions are ambiguous and so their absolute pixel values are having extremely few or zero interest. Morphological techniques probe a picture with a tiny low form or guide referred to as a structuring component. The structuring component is positioned the least bit potential locations within the image, and it is compared with the corresponding neighborhood of pixels. Some operations perform a test whether or not the component "fits" among the neighborhood, whereas others examine at whether or not its "hits" or intersects the neighborhood:

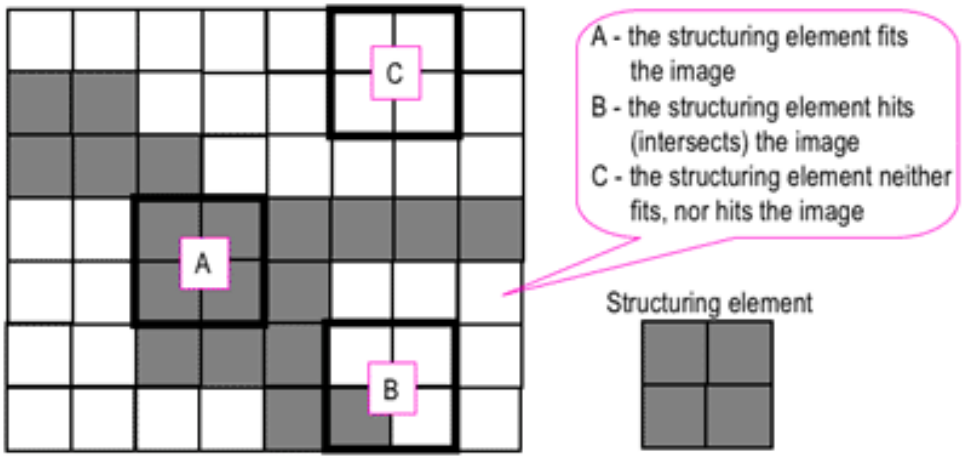


Figure 2.5: Probing of An Image with Structuring Element

Note. From 'Nick Efford. Digital Image Processing: A Practical Introduction Using JavaTM. Pearson Education, 2000.'

2.2.3 **Template Matching** [11] is a technique in digital image processing for locating tiny elements of a picture that match a guide image called template. It will be utilized in production line as a section of internal control [12], the simplest way to navigate a mobile mechanism [13], or as the simplest way to discover edges in pictures [14]. The most challenges within the template matching operations are; detection of non-rigid transformations, occlusion, background muddle, illumination and background changes, and scale changes [15].

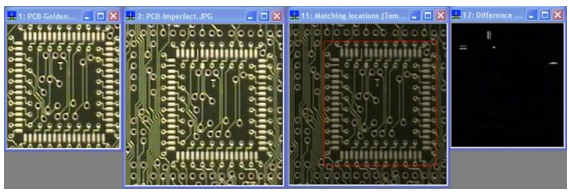


Figure 2.6: Template Matching Technique [23]

2.2.4 **Feature-based approach** depends on feature extraction process on a given picture such as, i.e. colors, shapes, textures, to relate within the target frame. By using Deep Neural Networks feature extraction networks such as VGG [16], ResNet, AlexNet Deep Convolutional Neural Networks we can easily score a high progress in this approach. We can describe the process of this approach as passing a given input with a fixed size to various of hidden layer and the output of each layer connected to a new next layer as input. Each layer's output includes some kind feature information about the image represented as vector. This approach is extraordinarily effective and has shown a very progressive result in solving a classification and detection problems. This technique is categorized as robust, sustain and is state of the art [17] it is capable of matching templates with non-rigid and enplane transformation [18][19] [20].

2.3 Dataset Preparation

In order to train and validate a custom designed model with our own application and requirements, we need a labeled dataset. A labeled dataset within the context of object detection images with a ground truth bounding box coordinates and labels.

2.4 Object Detection

Object detection is computer technology associated to computer vision and image processing that handle with detecting existence of semantic objects of certain kind (like humans, buildings, or cars)in digital images and videos, this kinds called the domain of detection include face detection and pedestrian detection Object detection has a lot of applications in many areas of computer vision, including image retrieval and video surveillance.

Every object class has unique characteristic that makes it easier for classifying this class for instance all triangles have 3 ribs only, these characteristics that object class detection use. For instance, when looking for the triangle can detect the 3 ribs and determine the kind of this rectangle by making a simple calculation uses the 3 angels, the same with square objects(ribs) that are perpendicular at corners and have equal side lengths are needed. This is the same idea for face recognition by detect the objects like eyes, nose and lips etc.…, a lot of characteristics of faces can be found also like skin color.

Machine learning-based approach or deep learning-based approaches is the fundamentals that object detection method based on, For Machine learning-based approach, it becomes indispensable to first define characteristics by using one of

methods below, then using a mechanism such as support vector machine (SVM) to do categorizing. on the other side deep learning mechanism is able to do end-to-end object recognition without specifically defining characteristics and are typically based on convolutional neural networks (CNN).

2.4.1 Machine Learning approaches

- Viola–Jones object detection framework based on Haar features
- Scale-invariant feature transform (SIFT)
- Histogram of oriented gradients (HOG) features [21]

2.4.2 Deep Learning approaches

- Region Proposals (R-CNN, Fast R-CNN, Faster R-CNN [9]):

  Faster R-CNN was created by scientists at Microsoft. It depends on R-CNN which utilized a multi-staged way to deal with object location. R-CNN utilized selective hunt to decide region proposals, pushed these through a grouping system and afterward utilized SVM to categorize the various areas and class.

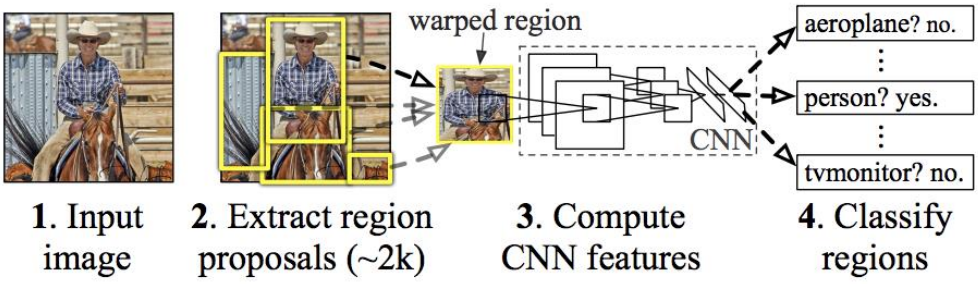


Figure 2.7: RCNN, Regions with CNN Features [9]

Faster R-CNN, like SSD, is an end to end state of the art approach during 2015. Rather than utilizing default bounding boxes, Faster R-CNN has a Region Proposal Network (RPN) to create a fixed arrangement of locales. The RPN utilizes the convolutional highlights from the picture

characterization network, empowering about sans cost region proposals. The RPN is executed as a fully convolutional network that its output is a prediction item bounding box and objectless scores at each position.

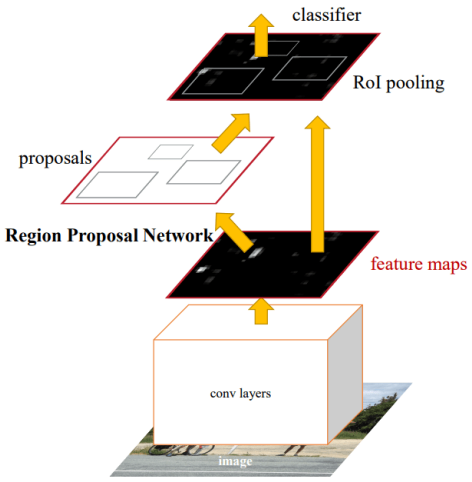


Figure 2.8: Faster RCNN Detection Process [9]

Note that the RPN has a comparative setup as the Single Shot Detector architecture network (for example it doesn't anticipate bounding boxes out of nowhere). The Region Proposals Network system works with sliding windows over the feature maps of each layer. At each sliding-window area or grapple, a lot of proposition are processed with different scales and angle proportions. Like SSD, the result of the RPN are 'balanced' bouncing boxes, in view of the stays. The various parts are consolidated in a solitary setup and are prepared either start to finish or in numerous stages (to improve security). Another understanding of the RPN is that it directs the systems 'consideration' to fascinating areas. The greater part of the use subtleties of Faster R-CNN are comparative as the ones for SSD. As far as crude mAP, Faster R-CNN commonly beats SSD, however it requires essentially increasingly computational power. A significant segment for the Fast-

RCNN finder, is the 'first_stage_anchor_generator' which characterizes the grapples created by the RPN. The steps in this segment characterizes the means of the sliding window. Note that particularly when endeavoring to recognize little items (if the walk is excessively enormous, we may miss them).

- Single Shot MultiBox Detector (SSD) [22]

- You Only Look Once (YOLO) [8]

  YOLO is an object detection system targeted for real-time processing.

  Class Prediction: Most classifiers assume output labels are mutually exclusive. It is true if the output are mutually exclusive object classes. Therefore, YOLO applies a SoftMax function to convert scores into probabilities that sum up to one. YOLOv3 uses multi-label classification. For example, the output labels may be “pedestrian” and “child” which are not non-exclusive. (the sum of output can be greater than 1 now.) YOLOv3 replaces the softmax function with independent logistic classifiers to calculate the likeliness of the input belongs to a specific label. Instead of using mean square error in calculating the classification loss, YOLOv3 uses binary cross-entropy loss for each label. This also reduces the computation complexity by avoiding the softmax function.

  Bounding box prediction & cost function calculation: YOLOv3 predicts an objectness score for each bounding box using logistic regression. YOLOv3 changes the way in calculating the cost function. If the bounding box prior (anchor) overlaps a ground truth object more than others, the corresponding

objectness score should be 1. For other priors with overlap greater than a predefined threshold (default 0.5), they incur no cost. Each ground truth object is associated with one boundary box prior only. If a bounding box prior is not assigned, it incurs no classification and localization lost, just confidence loss on objectness. We use tx and ty (instead of bx and by) to compute the loss.

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$

Figure 2.9: Yolo Bounding Box Equations [8]

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 2.10: YoloV3 Architecture [8]

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [3]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [6]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [4]	Inception-ResNet-v2 [19]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [18]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [13]	DarkNet-19 [13]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [9, 2]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [2]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [7]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [7]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

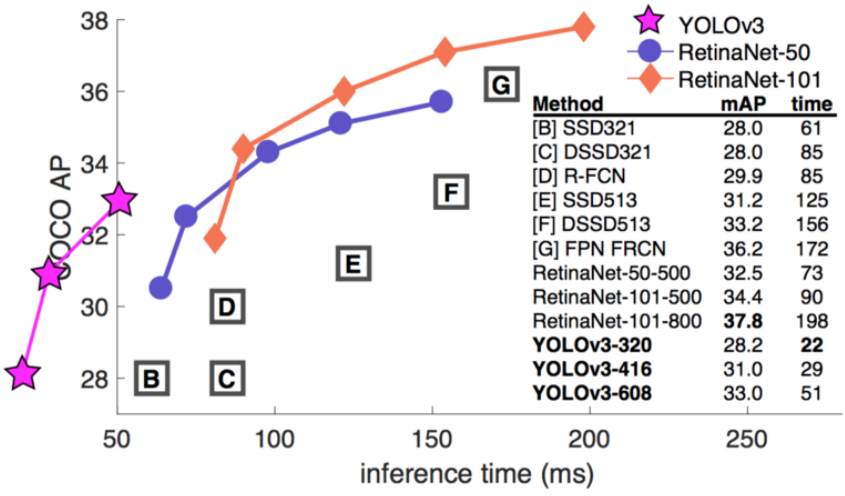


Figure 2.11: Current Object Detection Comparison [8]

2.5 CPU vs GPU

A central processing unit CPU is main and the most important part of any device for several reasons, the most important thing is the CPU is the only part which implement the instruction like control logical ,and input /output (I/O) operations, Simply, CPU function is to take input from any external source and use the parts of it which Control Unit (CU), Arithmetic logic unit (ALU), Cache, Dynamic Random Access Memory(DRAM). At 1970s the first version of CPU was done and designed by Intel, at this time the first version was designed by 1 core only and this was a simple one because the CPU can only operate one process at moment. after this and by a lot

of years and a lot of versions passed, this design developed into multi core CPUs that's mean at this time the CPU can operate 2 or more or more process at moment. Today the market have a lot of huge company that developed a lot of CPU that have a lot of cores , without huge different in the basic function and purpose of it (operating complex operations and computations), cores mostly useful for cases that needs Analysis of operations that have complex logic in code .

As we mentioned in CPU function that CPU performs a complex function but this maybe take a lot of time ,because the few number of parts like ALU that performs the calculations and arithmetic computations etc.., control unit that implement the microprocessor instructions, and cache memory, That's mean if the computer want to make a lot of operations of calculations in one moment it will be hard or will take long time even if it's a simple operations or calculations ,GPU comes with the solution of this problem by huge numbers of this logic cores that can implement operations or computations at the same moment.

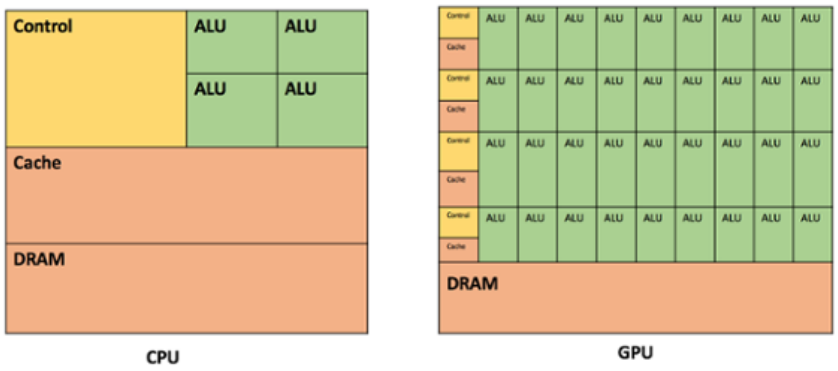


Figure 2.12: CPU vs GPU Architectures [24]

In 1970s and at the same time of starting gaming applications, GPU was already existing but Nivida wasn't released GeForce processor line of GPU, that is what make them more popular. They start by design GPU as particular graphical rendering workhorses of computer games, later the GPU improved to do the geometric

calculations quickly like transforming polygons and rotating verticals into different coordinate systems like 3D etc.…, Nvidia developed CUDA architecture and platform that allow to programmers to do operations in parallel out of code.

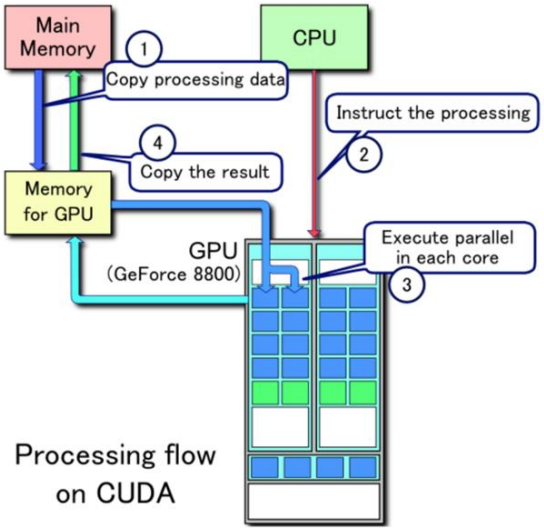


Figure 2.13: GPU Parallel Architecture [25]

Furthermore, a high percentage of this computations implicated matrix and vector operations, identical to type of mathematics is used in data science today. It doesn't take a long of time before engineers, developers and scientists makes researches in how GPU can be applied for non-graphical calculations

CHAPTER 3

METHODOLOGY

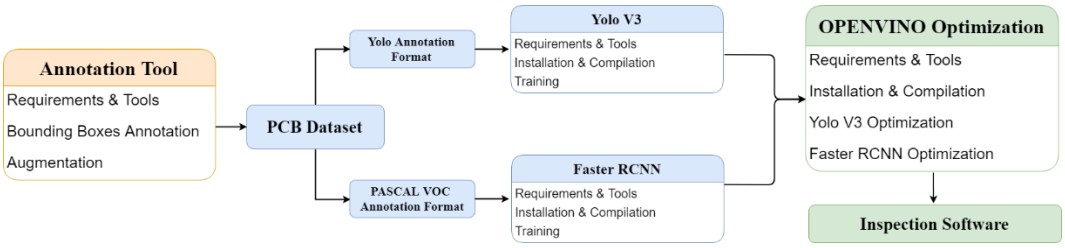


Figure 3.1: Block Diagram of Developing Our Software

3.1 Annotation Tool

Training deep neural networks requires data. Lots of data. And depending on the AI architecture at hand, that data isn't of much use if it isn't labeled, which takes time if we are not using the proper annotation tools. To ease the burden on data annotators and data scientists alike, we have developed a new program called UTeM

Annotator that's built to speed up annotation of image samples used to train computer vision algorithms. "Generally, there are many ways to annotate data, but using special tools (like UTeM Annotator) may help to speed up this process and augment our dataset."

3.1.1 Developing Tools and Requirements

Our Annotation software based on OpenCV and PyQt4 libraries and developed in python programming language:

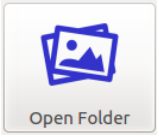
Table 3.1: Development Libraries Used for PCB Annotation Software

Library	Installation Command
Python2.7	sudo apt-get install python-pip
Numpy	sudo pip install numpy scipy
PyQT4	sudo apt-get install python-qt4
OpenCV 3.4.0	sudo pip install opencv-python

3.1.2 Bounding Box Annotation

In this section, I will clarify the function of each button on our tool and how the method that I used for implementing this functionality using python.

Table 3.2: Guide for Buttons of PCB Annotation Software

	Provides a dialog that allow users to select image or directories contacting our dataset images for starting of annotation process.
Definition	self.buttonopenFolder = QtGui.QToolButton(self) self.buttonopenFolder.setText(' Open Folder ') self.buttonopenFolder.setIcon(QtGui.QIcon('./res/image.png')) self.buttonopenFolder.setIconSize(QtCore.QSize(80,80)) self.buttonopenFolder.setCheckable(True) self.buttonopenFolder.setFixedSize(130,110) self.buttonopenFolder.setToolButtonStyle(QtCore.Qt.ToolButtonTextUnderIcon) self.buttonopenFolder.move(1710,15) self.buttonopenFolder.clicked.connect(self.openFloder)

Callback	def openFloder(self):
	dlg = QtGui.QFileDialog(self)
	dlg.setFileMode(QtGui.QFileDialog.AnyFile)
	dlg.setFilter("Image files (*.jpg)")
	dlg.show()
	if dlg.exec_():
	filenames = str(dlg.selectedFiles()[0])
	img = cv2.imread(filenames)
	annot_name=os.path.join(os.path.dirname(filenames),
	'{}.txt'.format(os.path.splitext(filenames)[0]))
	annot_data = []
	if os.path.isfile(annot_name):
	print('Loading existing annotation file')
	annot_data = parse_annotation_file(annot_name)
	self.newImage(filenames, img, annot_name, annot_data)
	self.redraw()

	Navigating throw images dataset that located at same chosen directory.
Definition	self.nextImageButton = QtGui.QToolButton(self) self.nextImageButton.setText('Previous \nImage') self.nextImageButton.setIcon(QtGui.QIcon('./res/left.png')) self.nextImageButton.setIconSize(QtCore.QSize(35,35)) self.nextImageButton.setFixedSize(60,90) self.nextImageButton.setCheckable(True) self.nextImageButton.setToolButtonStyle(QtCore.Qt.ToolButtonTextUnderIcon) self.nextImageButton.move(1710,135) self.nextImageButton.clicked.connect(self.fetchPreviousImage)

Callback	def fetchNextImage(self):
	if self.image_path is not None:
	path, _ = os.path.split(self.image_path)
	filesImage = glob.glob(path+'/*.jpg')
	indexOfImage = filesImage.index(self.image_path)
	if len(filesImage) > indexOfImage:
	img = cv2.imread(filesImage[indexOfImage+1])

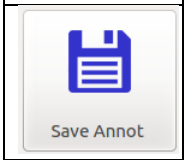
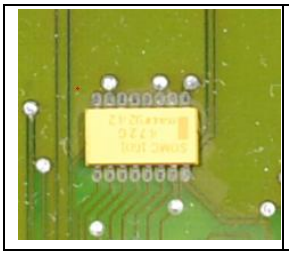
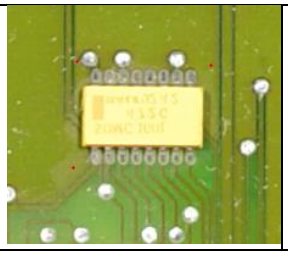
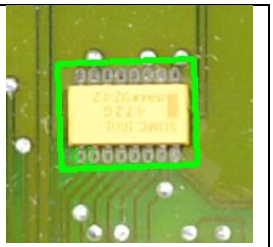
	annot_name = os.path.join(os.path.dirname(filesImage[indexOfImage+1]), '{}.txt'.format(os.path.splitext(filesImage[indexOfImage+1])[0])) annot_data = [] if os.path.isfile(annot_name): print('Loading existing annotation file') annot_data = parse_annotation_file(annot_name) self.newImage(filesImage[indexOfImage+1], img, annot_name, annot_data) self.redraw()
	Save all annotated bounding boxes to a text file
Definition	self.saveYoloFormat = QtGui.QToolButton(self) self.saveYoloFormat.setText(' Save Annot ') self.saveYoloFormat.setIcon(QtGui.QIcon('./res/save.png')) self.saveYoloFormat.setIconSize(QtCore.QSize(80,80)) self.saveYoloFormat.setCheckable(True) self.saveYoloFormat.setFixedSize(130,120) self.saveYoloFormat.setToolButtonStyle(QtCore.Qt.ToolButtonTextUnderIcon) self.saveYoloFormat.move(1710,250) self.saveYoloFormat.clicked.connect(self.saveAnnotation)
Callback	def saveAnnotation(self): print('Writing annotation data to "{}"'.format(self.annot_path)) write_annotation_file(self.annot, self.annot_path)

Table 3.3 explains step by step method of annotation for a new component. The user just need to click by his left click mouse at his desired (x1,y1), then the image will contain a red dot on his clicked coordinates, following the first click by another 3 rights clicks for (x2,y2), (x3,y3), (x4,y4) respectively. These coordinates represent 4 corners of our bounding box, and an angle will be generated automatically by our software. For deleting an existing annotation, just click on right click of the mouse and choose the desired bounding box.

Table 3.3: Guide for Drawing New Bounding Box

<table>
<tr><td>Step 1, pointing (x1,y1)</td><td>Step 2, pointing (x2,y2)</td><td>Step 3, pointing (x3,y3)</td><td>Step 4, pointing (x4,y4) with angle</td></tr>
<tr><td colspan="4">Annotation of object will be saved in below format:
(BBx_{center} , BBy_{center} , BBwidth, BBheight , BBangle and BBclass if available)</td></tr>
<tr><td>Defination</td><td colspan="3">self.label_image = MouseLabel(self)
self.scroll = QtGui.QScrollArea(self)
self.scroll.setWidget(self.label_image)
self.connect(self.label_image, QtCore.SIGNAL('clicked(int, int, int)'),
self.point_selected)</td></tr>
<tr><td>Callback</td><td colspan="3">def point_selected(self, x, y, button):
  if button != 1 and button != 2:
  return
  if button == 1:
  self.annot_coords[self.annot_idx][0] = x
  self.annot_coords[self.annot_idx][1] = y
  self.annot_idx += 1
  if self.annot_idx == 4:
  rect = cv2.minAreaRect(np.array(self.annot_coords))
  self.annot.append(Annot(rect))
  self.annot_idx = 0
  if button == 2:
  id = 0
  del_id = -1
  for an in self.annot:
  tmp = np.zeros(self.image.shape[:2], dtype=np.uint8)
  bp = cv2.cv.BoxPoints(an.rect)
  bp = np.int0(bp)
  cv2.drawContours(tmp, [bp], 0, (255), -1)
  if tmp[y, x] > 0:
  del_id = id
  break
  id += 1
  if del_id >= 0:
  del self.annot[del_id]
  self.redraw()</td></tr>
</table>

3.1.3 Data Augmentation

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset. Training deep learning neural network models on more data can result in more powerful models, and the augmentation techniques can create variations of the images that can improve the ability of the fit models to generalize what they have learned to new images. Since our

dataset contain a small amount of image that will cause a low detection accuracy, I have decided to apply data augmentation techniques to increase my training and validation dataset. Below are some data augmentation techniques that I have decided to use.

- Rotation (at 90 degrees)
- Salt and Pepper noise
- Brightness Condition

Table 3.4: Guide for Augmenting Our Dataset

<table>
<tr><td>  </td><td colspan="3">Preform a rotation augmentation by rotating the image and its annotation by the following degrees (90, 180, 270)</td></tr>
<tr><td>Definition</td><td colspan="3">self.rotateImage = QtGui.QToolButton(self)
self.rotateImage.setText(' Roation ')
self.rotateImage.setIcon(QtGui.QIcon('./res/rotate.png'))
self.rotateImage.setIconSize(QtCore.QSize(70,70))
self.rotateImage.setCheckable(True)
self.rotateImage.setFixedSize(130,95)
self.rotateImage.setToolButtonStyle(QtCore.Qt.ToolButtonTextUnderIcon)
self.rotateImage.move(1710,430)
self.rotateImage.clicked.connect(self.rotate3Image)</td></tr>
<tr><td>Callback</td><td colspan="3">def rotate3Image(self):
rotated1 = cv2.rotate(self.image, cv2.ROTATE_90_CLOCKWISE)
write_annotation_file_90(self.annot, (os.path.splitext(self.annot_path)[0]+'r90.txt'),self.image.shape[0])
cv2.imwrite((os.path.splitext(self.annot_path)[0]+'r90.jpg'),rotated1)
rotated2 = cv2.rotate(self.image, cv2.ROTATE_180)
write_annotation_file_180(self.annot, (os.path.splitext(self.annot_path)[0]+'r180.txt'),self.image.shape[0],self.image.shape[1])
cv2.imwrite((os.path.splitext(self.annot_path)[0]+'r180.jpg'),rotated2)
rotated3 = cv2.rotate(self.image, cv2.ROTATE_90_COUNTERCLOCKWISE)
write_annotation_file_270(self.annot, (os.path.splitext(self.annot_path)[0]+'r270.txt'),self.image.shape[1])
cv2.imwrite((os.path.splitext(self.annot_path)[0]+'r270.jpg'),rotated3)</td></tr>
<tr><td></td><td>90° Rotation</td><td>180° Rotation</td><td>270° Rotation</td></tr>
<tr><td>Equations (I: Image)</td><td>$X_{center} = I_{height} - Y_{center-old}$

$Y_{center} = X_{center-old}$</td><td>$X_{center} = I_{width} - X_{center-old}$

$Y_{center} = I_{height} - Y_{center-old}$</td><td>$X_{center} = Y_{center-old}$

$Y_{center} = I_{width} - X_{center-old}$</td></tr>
</table>

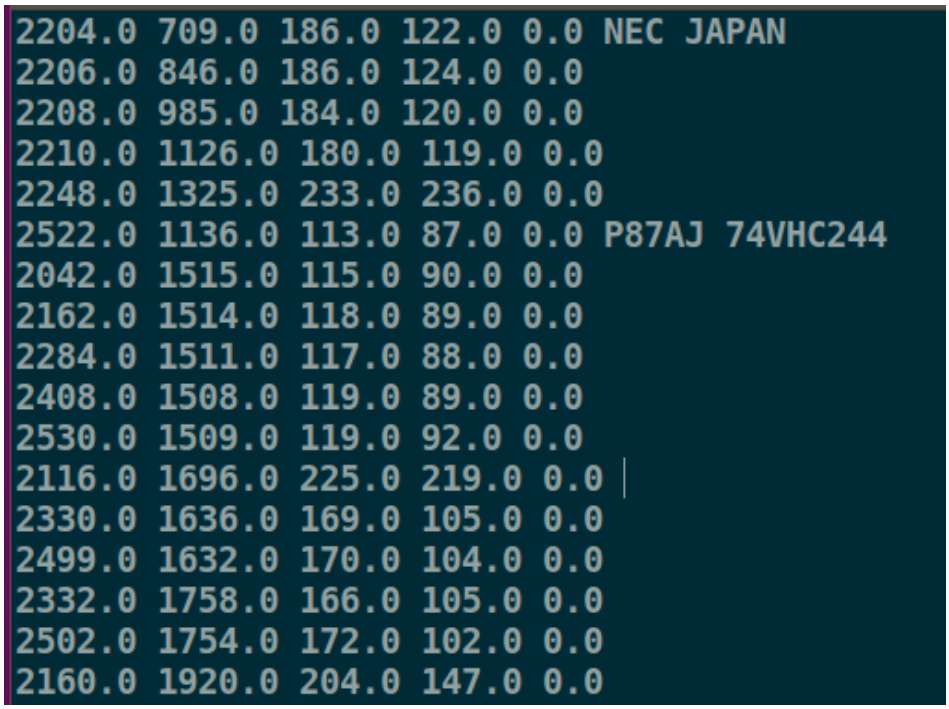
	Add noise by 2 intensities to the original image with different mean and standard division.
Definition	self.addnoise = QtGui.QToolButton(self) self.addnoise.setText(' Salt & Papper Noise ') self.addnoise.setIcon(QtGui.QIcon('./res/noise.png')) self.addnoise.setIconSize(QtCore.QSize(70,70)) self.addnoise.setCheckable(True) self.addnoise.setFixedSize(130,95) self.addnoise.setToolButtonStyle(QtCore.Qt.ToolButtonTextUnderIcon) self.addnoise.move(1710,630) self.addnoise.clicked.connect(self.addnoiseImage)
Callback	def addnoiseImage(self): img = self.image.copy() noiseImage = cv2.randn(img,(0,0,0),(5,5,5)) write_annotation_file(self.annot, (os.path.splitext(self.annot_path)[0]+'n1.txt')) cv2.imwrite((os.path.splitext(self.annot_path)[0]+'n1.jpg'),(self.image+noiseImage)) img2 = self.image.copy() noiseImage2 = cv2.randn(img2,(0,0,0),(10,10,10)) write_annotation_file(self.annot, (os.path.splitext(self.annot_path)[0]+'n2.txt')) cv2.imwrite((os.path.splitext(self.annot_path)[0]+'n2.jpg'),(self.image+noiseImage2))
	Augment original image by generating 2 different brighter images.
Definition	self.addBright = QtGui.QToolButton(self) self.addBright.setText(' Brightness ') self.addBright.setIcon(QtGui.QIcon('./res/bright.png')) self.addBright.setIconSize(QtCore.QSize(70,70)) self.addBright.setCheckable(True) self.addBright.setFixedSize(130,95) self.addBright.setToolButtonStyle(QtCore.Qt.ToolButtonTextUnderIcon) self.addBright.move(1710,730) self.addBright.clicked.connect(self.addBrightImage)
Callback	def addBrightImage(self): frameBright1 = increase_brightness(self.image, value=10) write_annotation_file(self.annot, (os.path.splitext(self.annot_path)[0]+'b1.txt')) cv2.imwrite((os.path.splitext(self.annot_path)[0]+'b1.jpg'),(frameBright1)) frameBright2 = increase_brightness(self.image, value=20) write_annotation_file(self.annot, (os.path.splitext(self.annot_path)[0]+'b2.txt')) cv2.imwrite((os.path.splitext(self.annot_path)[0]+'b2.jpg'),(frameBright2))

UTeM Annotation Tool executing using below line on a terminal using (Appendix A) script:

```
python UTeM_annotation.py
```

3.2 PCB Dataset

By using a public dataset that aim to facilitate research on computer-vision-based Printed Circuit Board (PCB) analysis, with a focus on recycling-related applications. The dataset contains originally from 748 images of PCBs from a recycling facility, captured under representative conditions using a professional DSLR camera but I have filtered these images to 163 images and augmented to be 1304 images total for training and validation. For all these images an accurate segmentation information for the depicted PCBs as well as bounding box information for all Integrated Circuit (IC) chips has been provided. Furthermore, textual information of the labels for a subset of IC samples has been provided. By including these different aspects of information, this dataset is useful for designing and testing a variety of methods for PCB analysis, from PCB classification over IC chip localization to the detection of specific chips.



$$\left(X_{center}, Y_{center}, BB_{width}, BB_{height}, BB_{angle}\ and\ class\ if\ available\right)$$

Figure 3.2: Sample of Original annotation format

A preprocessing has been done to generate the YOLO annotation format and PASCAL VOC format that is required for YOLO V3 and Faster-RCNN Training respectively.

3.2.1 YOLO Annotation

A python script has been developed for converting from original annotation format to Yolo annotation format using NumPy and OpenCV libraries. Yolo annotation format is text file for each image that contains all the object bounding boxes saved line by line as shown in Figure 3.3.

```
0 0.437703 0.529878 0.058036 0.088110
0 0.383726 0.145427 0.032062 0.065244
0 0.338271 0.153049 0.016640 0.025305
0 0.442370 0.143902 0.029221 0.045122
0 0.741883 0.370122 0.065138 0.098171
0 0.756899 0.707317 0.041599 0.049085
0 0.578328 0.254878 0.028003 0.029268
0 0.336851 0.328963 0.012581 0.027744
0 0.322037 0.381098 0.013596 0.021341
0 0.626015 0.281402 0.013190 0.018902
0 0.500609 0.448476 0.012175 0.023171
0 0.368304 0.490244 0.015016 0.019207
0 0.374797 0.530793 0.015016 0.017378
0 0.367086 0.621951 0.022119 0.040549
0 0.415787 0.111890 0.008320 0.018598
0 0.440341 0.111585 0.012175 0.012500
0 0.676136 0.194817 0.007914 0.018902
0 0.758117 0.159146 0.014205 0.017073
0 0.756494 0.198780 0.008523 0.020122
```

$(Class_{ID}, X_{center}, Y_{center}, BB_{width}, BB_{height})$

Figure 3.3: Sample of YoloV3 Annotation Format

```
def write_annotation_file(data, to , h, w):
    with open(to, 'w') as f:
        for d in data:
            f.write('0 {:.6f} {:.6f} {:.6f} {:.6f}\n'.format((d.rect[0][0]/w), (d.rect[0][1]/h), (d.rect[1][0]/w), (d.rect[1][1])/h))
```

Figure 3.4: Conversion Programming Equation of Original Annotation to Yolo Annotation

Where all Yolo annotation values are relative to the input image as specified at code above, where “d” represent each bounding box and

"f.write(…format)" have the conversion to yolo annotation format calculated using the below equations.

$$X_{center} = \frac{X_{absolte}}{Image_{width}} \qquad BB_{width} = \frac{BB_{width\_absolte}}{Image_{width}}$$

$$Y_{center} = \frac{Y_{absolte}}{Image_{height}} \qquad BB_{height} = \frac{X_{height\_absolte}}{Image_{height}}$$

For generating our new yolo annotation format, we need to allocate all our images and annotations at one folder and specify it in the code below:

```
filestext = glob.glob('/home/ibrahim/projects/FYP/Dataset/mva15/dataset/*.txt')
```

Figure 3.5: File Path of Original Dataset to Generate Yolo Ground Truth

Then we can execute our python script as shown below, where all new yolo annotation will be generated in our specified folder.

python converttoyolo.py (Appendix B)

3.2.2 PASCAL VOC Annotation

A sample XML annotation format called PASCAL VOC are required for Faster-RCNN training. An opensource script has been used for converting from YOLO annotation format to PASCAL VOC format.

wget_https://gist.github.com/goodhamgupta/7ca514458d24af980669b8b1c8bcdafd#file-yolo_to_voc-py

After configuring our dataset path variables in the downloaded script, we can execute our python script as the following:

python yolo_to_voc.py

PASCAL VOC annotation format uses XML file structure that includes our bounding boxes measurements represented by different values as shown below:

$$(X_{min} , Y_{min} , X_{max}, Y_{max})$$

calculated using the below equations.

$$X_{min} = X_{center} - \frac{BB_{width}}{2} \qquad X_{max} = X_{center} + \frac{BB_{width}}{2}$$

$$Y_{min} = Y_{center} - \frac{BB_{height}}{2} \qquad Y_{max} = Y_{center} + \frac{BB_{height}}{2}$$

We will be able to get a new folder that include all the annotation in PASCAL VOC format, as shown in Figure 3.6 Sample:

```
<?xml version="1.0"?>
- <annotations>
    <filename>pcb1.jpg</filename>
    <folder>Final_Dataset_Yolo</folder>
  - <size>
      <width>4928</width>
      <height>3280</height>
      <depth>3</depth>
    </size>
  - <object>
      <name>IC</name>
      <pose>Unspecified</pose>
      <truncated>0</truncated>
      <difficult>0</difficult>
    - <bndbox>
        <xmin>2013.99968</xmin>
        <ymin>1593.49944</ymin>
        <xmax>2300.001088</xmax>
        <ymax>1882.50024</ymax>
      </bndbox>
    </object>
```

Figure 3.6: Sample of PASCAL VOC Annotation Format

3.3 Hardware and System

The machine used in this experiment is Intel Xeon E5-2620 v3 processor with 47GB RAM. The GPU card used is NVIDIA GeForce GTX-1080 TI with driver version 390.87. The operating system is 64-bits Ubuntu 16.04 LTS. The TensorFlow used is version 1.12.0 precompiled version installed via PIP library, Protobuf 3.0.0 and OpenCV version 3.4.0 with CUDA 9.0 and cuDNN v7.0.

3.4 YOLO V3

3.4.1 Compilation and Testing

We need to download the source code of Yolo V3 using below line. The line should be executed at ubuntu terminal and our computer should have "Git" library installed:

```
git clone https://github.com/AlexeyAB/darknet.git
```

In darknet directory, we can find Makefile that we should edit it according to our choice, I have edit to values as shown on Figure 3.7.

```
GPU=1
CUDNN=1
OPENCV=1
OPENMP=0
DEBUG=1
```

Figure 3.7: YoloV3 Makefile Configuration

1st line is identifiying if we want to comile Yolo using GPU or CPU only, if it equal to one, will be using GPU during the training and infearnce. 2nd identifies the usage of CUDNN library from Nvidia. 3rd and 4th identify the usage of OPENCV and OPENMP library. Finally last line enable the debug mode.

```
cd darknet ; sudo make
```

Compile YoloV3 by chosen configuration. Then we are required to download a pretrainied model for testing a simple object detection from our webcamera using standard configuration of yolov3 with an input imagesize of 416x416 and 80 classes as specified at `data/coco.names` file.

```
wget https://pjreddie.com/media/files/yolov3.weights
```

```
./darknet detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights
```

3.4.2 Training Process

Create file yolo-obj.cfg with the same content as in yolov3.cfg, then we change line batch to batch=128, line subdivisions to subdivisions=32, line max_batches to max_batches=50200, line steps to steps=40000,45000, line classes=80 to classes=1 in each of 3 [yolo]-layers, [filters=255] to filters=(classes + 5)*3 = (1+5)*3=18 in the 3 [convolutional] before each [yolo] layer.

Then we create file obj.names in darkent\data\ directory , with objects names – “IC”. Then we create file obj.data in the directory darknet\data\, containing number of classes, training and validation dataset paths with exporting output file path.

```
classes= 1
train  = ./train.txt
valid  = ./valid.txt
names = ./obj.names
backup = ./backup/
```

Figure 3.8: YoloV3 obj.data Training File

Darknet requires 2 text file (train.txt and valid.txt) with filenames of our images, each filename in new line, with an absoulte path as shown in Figure 3.9:

```
/home/ubuntu/TF_API/FYP/Final_Dataset_Yolo/pcb2r270.jpg
/home/ubuntu/TF_API/FYP/Final_Dataset_Yolo/pcb120n2.jpg
/home/ubuntu/TF_API/FYP/Final_Dataset_Yolo/pcb94.jpg
/home/ubuntu/TF_API/FYP/Final_Dataset_Yolo/pcb67n2.jpg
/home/ubuntu/TF_API/FYP/Final_Dataset_Yolo/pcb154r270.jpg
/home/ubuntu/TF_API/FYP/Final_Dataset_Yolo/pcb59n2.jpg
/home/ubuntu/TF_API/FYP/Final_Dataset_Yolo/pcb55b1.jpg
```

Figure 3.9: YoloV3 Training Dataset Path (train.txt)

We can generate these 2 text file with dividing our dataset to (80% training and 20% validation) by executing the below command:

python process.py (Appendix C)

For training I use convolutional weights that are pre-trained on ImageNet. I use weights from the darknet53 model. That can be downloaded by the following command line:

wget https://pjreddie.com/media/files/darknet53.conv.74

Now we can start training by running the below command from darknet root directory:

./darknet detector train cfg/obj.data cfg/yolo-obj.cfg darknet53.conv.74 -map > log.txt

3.5 Faster RCNN

3.5.1 Compilation and Installation

We are going to use tesorflow object detection API for our training and inference of Faster RCNN, so after downloading tensorflow

object detection API, we are required to install the libraries listed in Table 3.5 to complete the installation

```
git clone https://github.com/tensorflow/models.git
```

Table 3.5: Libraries Used in Training of Faster FRCNN

Library	Installation Command
Protobuf	sudo apt-get install protobuf-compiler
Python-tk	sudo apt-get install python-tk
Pillow 1.0	sudo apt-get python-pil
lxml	sudo apt-get python-lxml
Jupyter notebook	sudo pip install --user jupyter
Matplotlib	sudo pip install --user matplotlib
Cython	sudo pip install --user Cython
contextlib2	sudo pip install --user contextlib2

We are required to download and install the 3.0 release of protoc, then unzip the file.

```
# From tensorflow/models/research/
wget -O protobuf.zip https://github.com/google/protobuf/releases/download/v3.0.0/protoc-3.0.0-linux-x86_64.zip
unzip protobuf.zip
```

then we run the compilation process again but using the downloaded version of protoc.

```
# From tensorflow/models/research/
./bin/protoc object_detection/protos/*.proto --python_out=.
```

When running locally, the tensorflow/models/research/ and slim directories should be appended to PYTHONPATH. This can be done by running the following from tensorflow/models/research/:

```
# From tensorflow/models/research/
export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim
```

3.5.2 Training Process

To run our training, we are required to generate a tf _record from our PASCAL VOC training and validation dataset and prepare pipeline configuration for Faster RCNN.

Tensorflow Object Detection API reads data using the TFRecord file format. A sample script (create_pascal_tf_record.py) are provided to convert from the PASCAL VOC dataset to TFRecords. Firstly, we need to download the PASCAL VOC general dataset by executing the following commands:

```
# From tensorflow/models/research/
Wget
http://host.robots.ox.ac.uk/pascal/VOC/voc2012/VOCtrainval_11
-May-2012.tar
tar -xvf VOCtrainval_11-May-2012.tar
```

Secondly, we shall replace the JPG Image by our dataset image and replace the annotation image by our annotation images in the following paths respectively ‘./VOCdevkit/VOC2012/JPEGImages’ and ‘./VOCdevkit/VOC2012/Annotations’. Then we are required to copy ‘train.txt and vaild.txt’ that has been generated for yolo

training and paste them at ‘./VOCdevkit/VOC2012/ImageSets/Main/’. Then we are required to create our object detection label map file at ‘object_detection/data/pascal_label_map.pbtxt’ which has our class ID and name “ID = 0, Name = IC”. Lastly, we can generate our training and testing tf_record by running below commands, they will be generated at research root directory:

```
python object_detection/dataset_tools/create_pascal_tf_record.py --label_map_path=object_detection/data/frcnn_label_map.pbtxt --data_dir=VOCdevkit --year=VOC2012 --set=train --output_path=frcnn_train.record
```

```
python object_detection/dataset_tools/create_pascal_tf_record.py --label_map_path=object_detection/data/frcnn_label_map.pbtxt --data_dir=VOCdevkit --year=VOC2012 --set=test --output_path=frcnn_test.record
```

The Tensorflow Object Detection API uses protobuf files to configure the training and evaluation process. The schema for the training pipeline can be found in object_detection/protos/pipeline.proto. At a high level, the config file is split into 5 parts:

- The model configuration. This defines what type of model will be trained (ie. meta-architecture, feature extractor). As shown in Figure 3.10:

```
model {
  faster_rcnn {
    num_classes: 1
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 600
        max_dimension: 1024
      }
    }
    feature_extractor {
      type: 'faster_rcnn_inception_v2'
      first_stage_features_stride: 16
    }
```

Figure 3.10: Faster RCNN Model Configuration

- The train_config, which decides what parameters should be used to train model parameters (ie. SGD parameters, input preprocessing and feature extractor initialization values).

```
train_config: {
  batch_size: 1
  optimizer {
    momentum_optimizer: {
      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
    }
```

Figure 3.11: Faster RCNN Train Configuration

- The eval_config, which determines what set of metrics will be reported for evaluation.

```
eval_config: {
  num_examples: 8000
  # Note: The below line limits the evaluation process to 10 evaluations.
  # Remove the below line to evaluate indefinitely.
  max_evals: 10
}
```

Figure 3.12: Faster RCNN Evaluation Configuration

- The train_input_config, which defines what dataset the model should be trained on.

Figure 3.13: Faster RCNN Training Input Reader

- The eval_input_config, which defines what dataset the model will be evaluated on. Typically, this should be different than the training input dataset.

Figure 3.14: Faster RCNN Evaluation Input Reader

After our preparation for training and testing TF_Records and our network configuration, we will download a pretrained model and start a transfer learning process by executing the following commands:

```
#From model/research
wget
http://download.tensorflow.org/models/object_detection/faster_rcnn
_inception_v2_coco_2018_01_28.tar.gz

tar -xvf faster_rcnn_inception_v2_coco_2018_01_28.tar.gz

python object_detection/model_main.py --
pipeline_config_path=/home/ubuntu/TF_API/FYP/models/res
earch/object_detection/samples/configs/ibrahim_faster_rcnn_i
nception_v2_coco.config --model_dir=./outputFasterRCNN
--num_train_steps=50000 --
sample_1_of_n_eval_examples=1 --alsologtostderr
```

3.5.3 Exporting a trained model for inference

After our model has been trained, we should export it to a Tensorflow graph proto. A checkpoint will typically consist of three files:

- model.ckpt-${CHECKPOINT_NUMBER}.data-00000-of-00001
- model.ckpt-${CHECKPOINT_NUMBER}.index
- model.ckpt-${CHECKPOINT_NUMBER}.meta

After we have identified a candidate checkpoint to export, we run the following command from tensorflow/models/research:

```
python object_detection/export_inference_graph.py --
input_type=image_tensor --
pipeline_config_path=/home/ubuntu/TF_API/FYP/models/research/
object_detection/samples/configs/ibrahim_faster_rcnn_inception_v
2_coco.config --
trained_checkpoint_prefix=./outputFasterRCNN/model.ckpt-50200
--output_directory=./outputFasterRCNN
```

After export, we should see the directory ${EXPORT_DIR} containing the following:

- saved_model/, a directory containing the saved model format of the exported model.
- frozen_inference_graph.pb, the frozen graph format of the exported model
- model.ckpt.*, the model checkpoints used for exporting
- checkpoint, a file specifying to restore included checkpoint files
- pipeline.config, pipeline config file for the exported model

3.6 Determine the accuracy of person detection with mean average precision (mAP)

The obtained result of IC detection provides a bounding box to indicate the location of the detected IC in the image. Mean Average Precision (mAP) is commonly used by the object detection research community to indicate how well the bounding box predicted by the trained object detection model overlaps with the labeled ground truth bounding box. The mAP is calculated as

$$mAP = \frac{\sum AP\ of\ \ all\ classes}{Number\ of\ \ classes}$$

The Average Precision (AP) is calculated as

$$AP = recall \times precision$$

The recall is calculated as

$$recall = \frac{no.\ of\ \ TP}{no.\ of\ \ TP + no.\ of\ \ FN}$$

where: TP = True Positive

FN = False Negative

The precision is calculated as

$$precision = \frac{no.\ of\ \ TP}{no.\ of\ \ TP + no.\ of\ \ FP}$$

where: TP = True Positive

FP = False Positive

3.7 OPENVINO

3.7.1 Installation

Download latest OpenVINO toolkit throw this link:

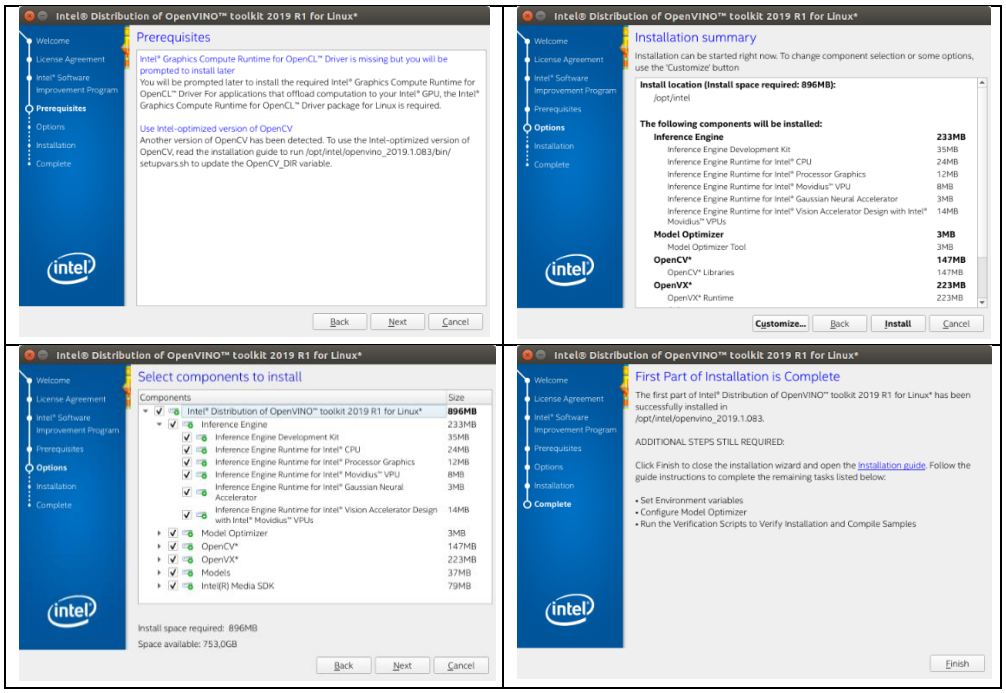
https://software.intel.com/en-us/openvino-toolkit/choose-download

We open a command prompt terminal window and unpack the .tgz file:

```
tar -xvzf l_openvino_toolkit_p_<version>.tgz
cd l_openvino_toolkit_p_<version>
```

Start installation by running: `sudo ./install_GUI.sh` and following installation instruction shown in Table 3.6

Table 3.6: OPENVINO Installation Guide



Install External Software Dependencies, these dependencies are required for:

- Intel-optimized OpenCV
- Deep Learning Inference Engine

- Deep Learning Model Optimizer tools

Change to the install_dependencies directory:

```
cd /opt/intel/openvino/install_dependencies
```

Run a script to download and install the external software dependencies:

```
sudo -E ./install_openvino_dependencies.sh
source /opt/intel/openvino/bin/setupvars.sh
```

Configure the Model Optimizer

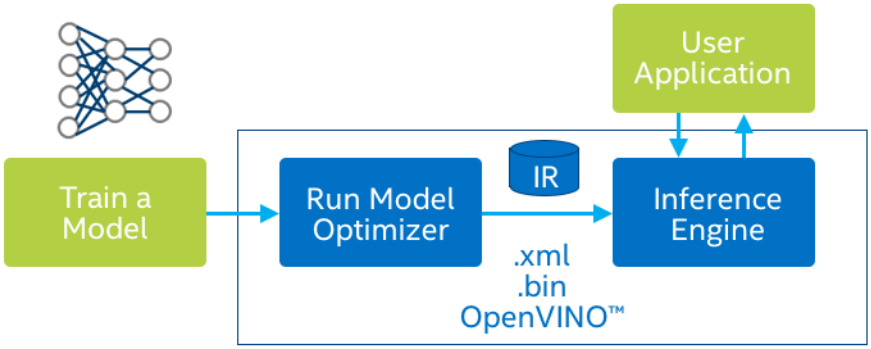


Figure 3.15: OPENVINO Optimization Process [27]

The Model Optimizer is a key component of the Intel Distribution of OpenVINO toolkit. We cannot perform inference on our trained model without running the model through the Model Optimizer. When we run a pre-trained model through the Model Optimizer, our output is an Intermediate Representation (IR) of the network. The Intermediate Representation is a pair of files that describe the whole model:

- .xml: Describes the network topology
- .bin: Contains the weights and biases binary data

Change directory to Model Optimizer prerequisites:

```
cd /opt/intel/openvino/deployment_tools/model_optimizer/install_prerequisites
```

Run the script to configure the Model Optimizer for Caffe, TensorFlow, MXNet, Kaldi*, and ONNX:

```
sudo ./install_prerequisites.sh
```

3.7.2 Yolo Optimization

All YOLO* models are originally implemented in the DarkNet* framework and consist of two files:

.cfg file with model configurations

.weights file with model weights

Originally, YOLOv3 model includes feature extractor called Darknet-53 with three branches at the end that make detections at three different scales. These branches must end with the YOLO Region layer. Region layer was first introduced in the DarkNet framework. Other frameworks, including TensorFlow, do not have the Region implemented as a single layer, so every author of public YOLOv3 model creates it using simple layers. This badly affects performance. For this reason, the main idea of YOLOv3 model conversion to IR is to cut off these custom Region-like parts of the model and complete the model with the Region layers where required. So, our first step to convert our YOLOv3 weight to tensorflow weight representation ".pb", by dumping a YOLOv3 TensorFlow implementation

Clone the repository:

```
git clone https://github.com/mystic123/tensorflow-yolo-v3.git
cd tensorflow-yolo-v3
```

Then we use our obj.name files that include our classes and our generated weights from training as below:

```
python3 convert_weights_pb.py --class_names obj.names --data_format NHWC --weights_file yolo-obj_15000.weights
```

Now we can convert YOLOv3 TensorFlow Model to the IR, but we need to edit our yolov3.json file allocated at <OPENVINO_INSTALL_DIR>/deployment_tools/model_optimizer/extensions/front/tf to follow our configuration and number of classes as shown in Figure 3.16.

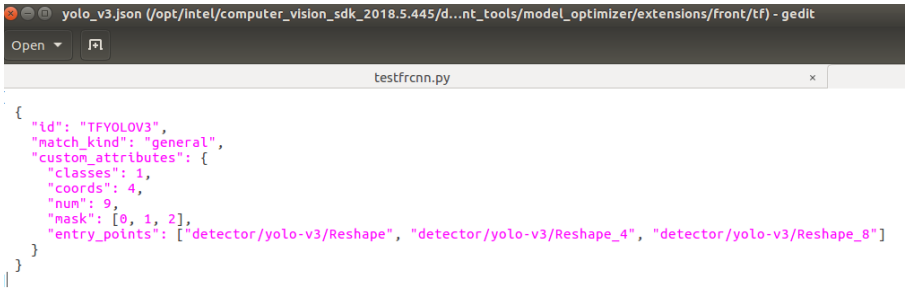


Figure 3.16: YoloV3 Custom Layer Optimization Configuration

To generate the IR of the YOLOv3 TensorFlow model, run:

```
python3 mo_tf.py --input_model /home/ubuntu/TF_API/FYP/darkflow/YOLO-OpenVINO-master/tensorflow-yolo-v3/frozen_darknet_yolov3_model.pb --tensorflow_use_custom_operations_config /opt/intel/computer_vision_sdk_2018.5.445/deployment_tools/model_optimizer/extensions/front/tf/yolo_v3.json --batch 1
```

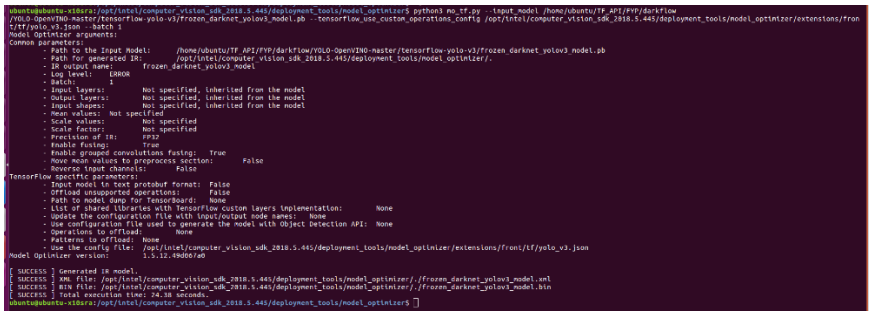


Figure 3.17: YoloV3 Optimization Output

3.7.3 Faster RCNN Optimization

Since we have generated our Faster RCNN frozen graph format of the exported model, we can convert our TensorFlow Model to the IR, but we need to edit our faster_rcnn_support_api_v.1.7.json file allocated at <OPENVINO_INSTALL_DIR>/deployment_tools/-model_optimizer/extensions/front/tf to follow our configuration and custom layer topology of Faster RCNN by adding following objects to the end of the file and save it as a new filename as faster_rcnn_support_api_v.1.13.json.

```
{
    "custom_attributes": {
        "outputs": "detection_boxes,detection_scores,num_detections"
    },
    "id": "ObjectDetectionAPIOutputReplacement",
    "match_kind": "general"
},
{
    "custom_attributes":
    {
        "replacements": [["mul/y", "first_stage_max_proposals"]]
    },
    "id": "ObjectDetectionAPIConstValueOverride",
    "match_kind": "general"
}
```

Figure 3.18: Faster RCNN Custom Layer Optimization Configuration

To generate the IR of the YOLOv3 TensorFlow model, we run below command from <OPENVINO_INSTALL_DIR>/-deployment_tools/model_optimizer/ :

```
python3 mo_tf.py --input_model /home/ubuntu/TF_API/FYP/models/research/outputFasterRCNN/frozen_inference_graph.pb --tensorflow_use_custom_operations_config /opt/intel/computer_vision_sdk_2018.5.445/deployment_tools/model_optimizer/extensions/front/tf/faster_rcnn_support_api_v1.13.json --tensorflow_object_detection_api_pipeline_config /home/ubuntu/TF_API/FYP/models/research/object_detection/samples/configs/ibrahim_faster_rcnn_inception_v2_coco.config --batch 1
```

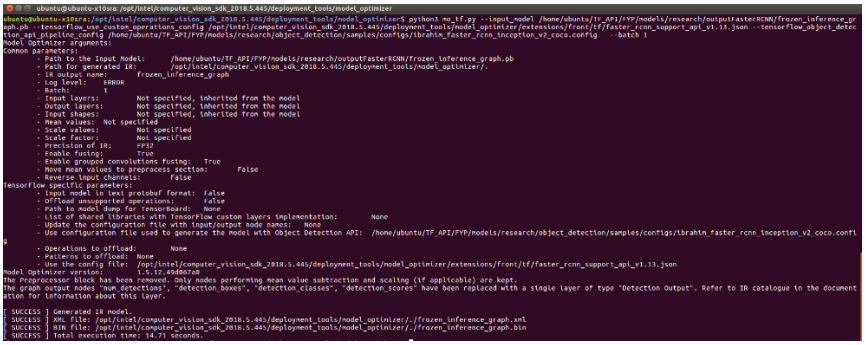


Figure 3.19: Faster RCNN Optimization Output

CHAPTER 4

RESULT AND DISCUSSION

4.1 Annotation Tool

By loading a single JPG or PNG image format, our tool will be able to load any exciting annotation file (.txt) with same file name of the image and visualize all annotated bounding boxes on our image. By performing our 3 augmentation techniques, we will be able to generate 7 images from our original image as shown in Table 4.1.

Table 4.1: Augmentation Techniques and Annotation Files name

Augmentation Technique	Number of Generated Image
Rotation	3 Images (filename+r90, r180, r270.txt & .jpg)
Brightness Level	2 Images (filename+b1, b2.txt & .jpg)
Noise Level	2 Images (filename+n1, n2.txt & .jpg)

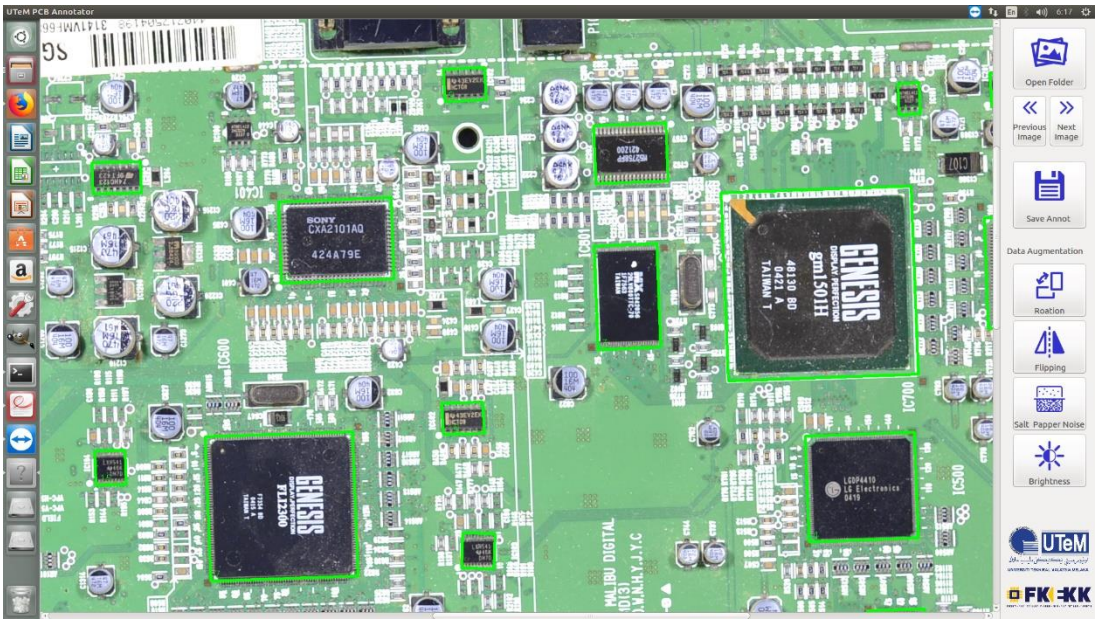


Figure 4.1: UTeM Developed Annotation Tool

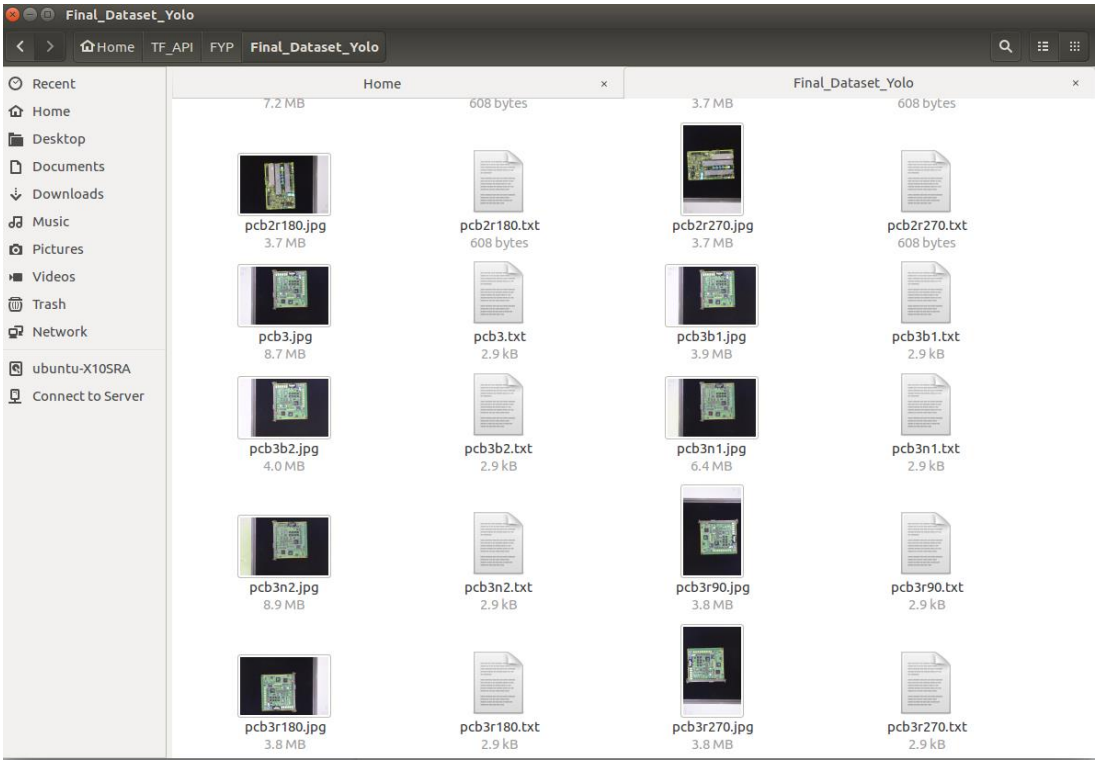
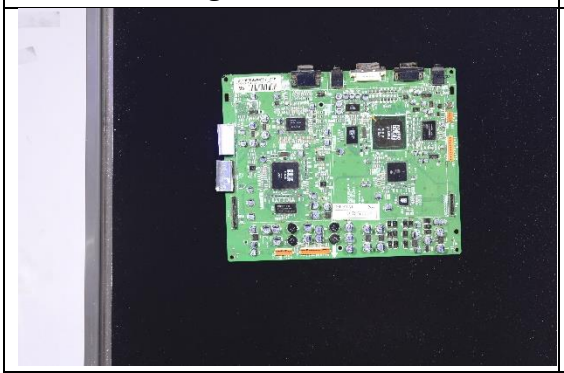
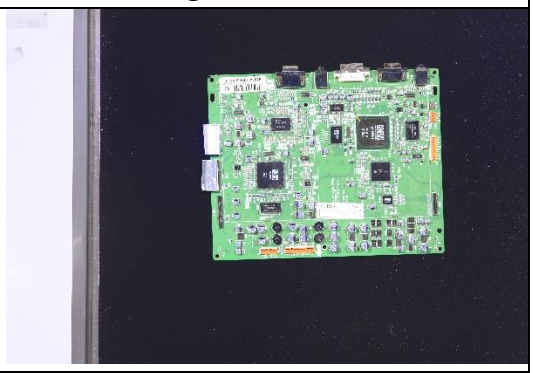
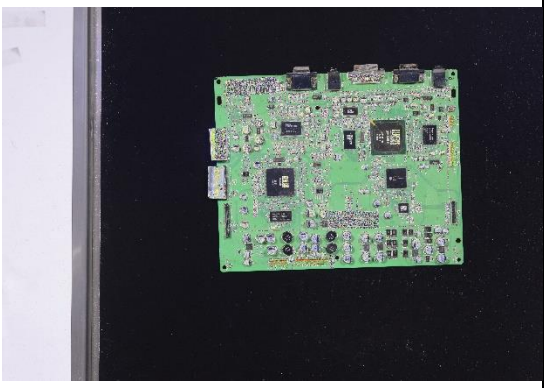
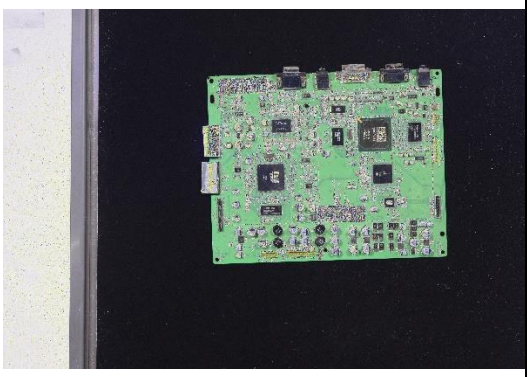
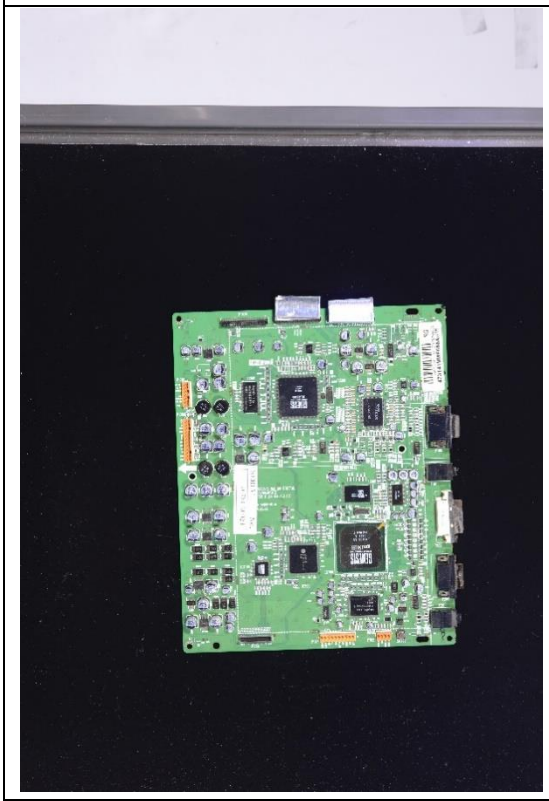
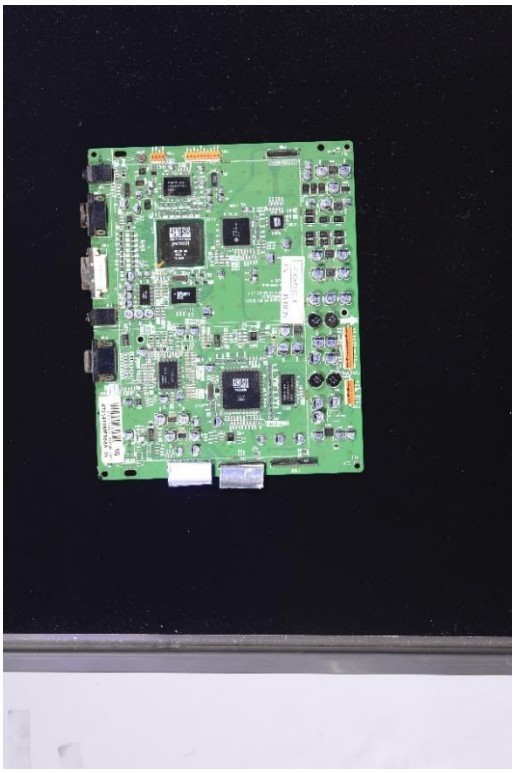
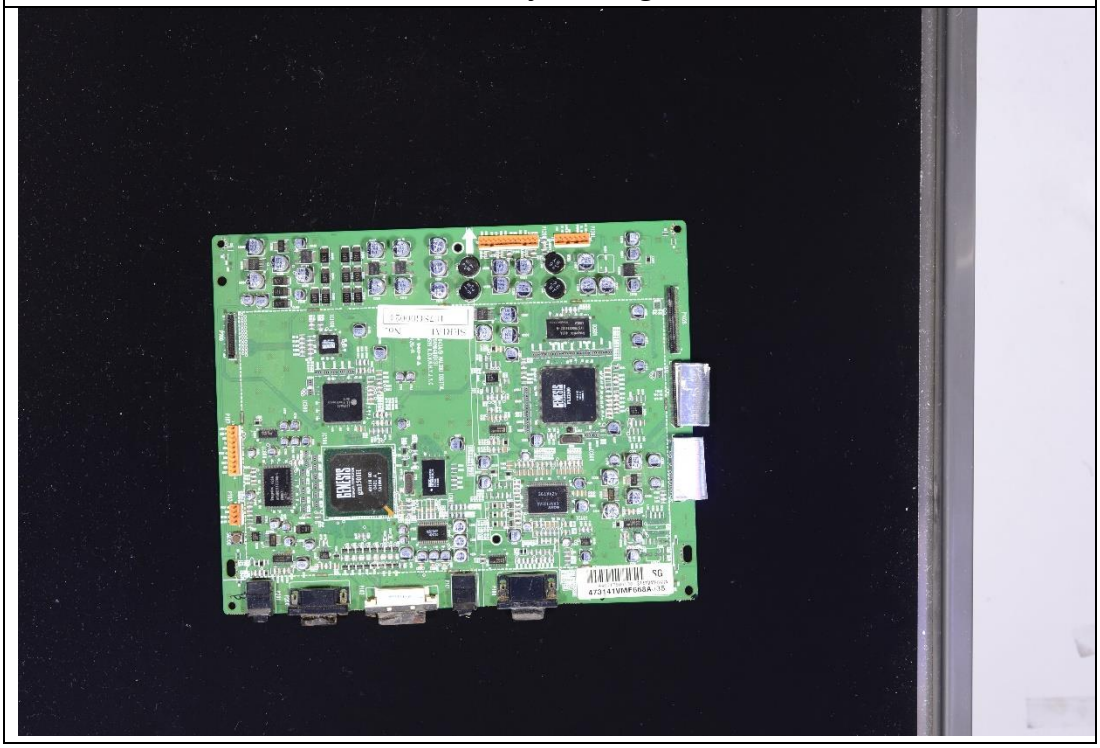


Figure 4.2: Output Ground Truth using Available Augmentation Techniques

Table 4.2: Sample of Augmentation Output

Brightness level 1	Brightness level 2
	
Noise level 1	Noise level 2
	
Rotation by 90 degrees	Rotation by 270 degrees
	

Rotation by 180 degrees


4.2 PCB Dataset Preparation

According to our annotation tool output formatting of ground truth bounding boxes, we are required to convert our output to two different formats (YoloV3 & PASCAL VOC) to start our training as discussed in the methodology. An output sample of different ground truth format are presented in Table 4.3.

Table 4.3: Training Annotation Formats

Our Annotation format	Yolo V3 foramt	PASCAL VOC Format
2204.0 709.0 186.0 122.0 0.0 NEC JA 2206.0 846.0 186.0 124.0 0.0 2208.0 985.0 184.0 120.0 0.0 2210.0 1126.0 180.0 119.0 0.0 2248.0 1325.0 233.0 236.0 0.0 2522.0 1136.0 113.0 87.0 0.0 P87AJ 2042.0 1515.0 115.0 90.0 0.0 2162.0 1514.0 118.0 89.0 0.0 2284.0 1511.0 117.0 88.0 0.0 2408.0 1508.0 119.0 89.0 0.0 2530.0 1509.0 119.0 92.0 0.0 2116.0 1696.0 225.0 219.0 0.0 2330.0 1636.0 169.0 105.0 0.0 2499.0 1632.0 170.0 104.0 0.0 2332.0 1758.0 166.0 105.0 0.0 2502.0 1754.0 172.0 102.0 0.0 2160.0 1920.0 204.0 147.0 0.0	0 0.437703 0.529878 0.058036 0.088110 0 0.383726 0.145427 0.032062 0.065244 0 0.338271 0.153049 0.016640 0.025305 0 0.442370 0.143902 0.029221 0.045122 0 0.741883 0.370122 0.065138 0.098171 0 0.756899 0.707317 0.041599 0.049085 0 0.578328 0.254878 0.028003 0.029268 0 0.336851 0.328963 0.012581 0.027744 0 0.322037 0.381098 0.013596 0.021341 0 0.626015 0.281402 0.013190 0.018902 0 0.500609 0.448476 0.012175 0.023171 0 0.368304 0.490244 0.015016 0.019207 0 0.374797 0.530793 0.015016 0.017378 0 0.367086 0.621951 0.022119 0.040549 0 0.415787 0.111890 0.008320 0.018598 0 0.440341 0.111585 0.012175 0.012500 0 0.676136 0.194817 0.007914 0.018902 0 0.758117 0.159146 0.014205 0.017073 0 0.756494 0.198780 0.008523 0.020122	<?xml version="1.0"?> <annotations> <filename>pcb1.jpg</filename> <folder>Final_Dataset_Yolo</folder> <size> <width>4928</width> <height>3280</height> <depth>3</depth> </size> <object> <name>IC</name> <pose>Unspecified</pose> <truncated>0</truncated> <difficult>0</difficult> <bndbox> <xmin>2013.99968</xmin> <ymin>1593.49944</ymin> <xmax>2300.001088</xmax> <ymax>1882.50024</ymax> </bndbox> </object>
(X_{center} , Y_{center} , BB_{width}, BB_{height} , BB_{angle} *and class if available*)	($Class_{ID}$, X_{center} , Y_{center} , BB_{width}, BB_{height})	(X_{min} , Y_{min} , X_{max}, Y_{max})

4.3 Performance analysis for convolutional object detector in IC allocation using YoloV3 and Faster RCNN

The performance analysis for convolutional object detector in IC allocation with YoloV3 and Faster RCNN consists of: (i) number of training iterations versus mAP and loss (ii) number or training iteration versus training time and memory usage (iii) inference time using different hardware accelerator.

4.3.1 Number of training iterations versus mAP and loss

During the training for YoloV3 and Faster RCNN models on PCB IC component dataset [26], the checkpoint file is saved every 100 steps for the first 1000 iterations, and saved each 1000 iterations for the rest of the training, this could help us to study how the mean average precision (mAP) and loss are changed across the time and recover our weight in case of any failure happens to our PC during training. Furthermore, this could provide an insight on how many training iterations is needed for each model to converge and show a clear figure in case of training overfitting happened. The training is stopped at 50,200 iterations as there is little improvement on the mAP for YoloV3 and Faster RCNN models and no much decrease in loss score of our models. The final mAP and loss at last iteration for both models is shown in Table 4.4. The mAP and loss versus the number of iterations for Faster RCNN and Yolo V3 is shown in Figure 4.3, Figure 4.4 and Figure 4.5 respectively. According to both figures, we can analyze a periodic increase of mAP score over training iteration during Faster RCNN training and it shows unfluctuating scores that can be improved by extending the training time, meanwhile a huge drop of loss across the 1000 iterations and periodic decrease over the rest of training iterations. On the other hand, we can observe that YoloV3 Model have reach to its highest mAP scores (red line) after 5200 iterations only and keep

fluctuating between 0.89 and 0.90 mAP score without much increase in mAP across the rest of training iterations, meanwhile the loss has dramatically decrease over first 5200 iterations and preserve a slight decrease over the rest of training iterations.

Table 4.4: Faster RCNN and YoloV3 mAP and Loss Scores

IC detection models	mAP (50,200 iterations)	Loss (50,200 iterations)
Faster RCNN	0.43	0.32
Yolo V3	0.89	0.23

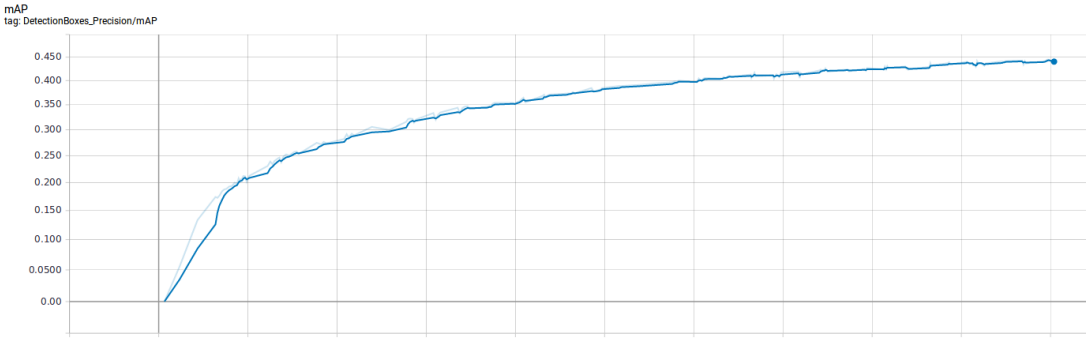


Figure 4.3: Faster RCNN mAP Training Graph

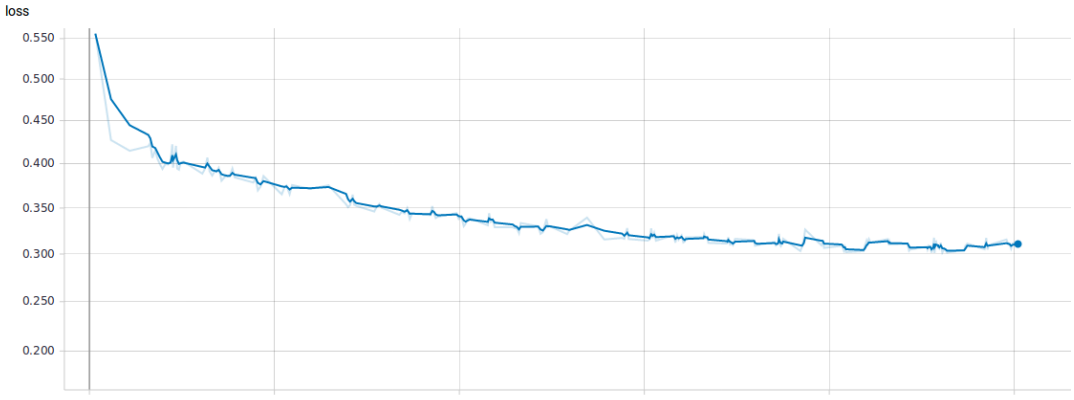


Figure 4.4: Faster RCNN Loss Training Graph

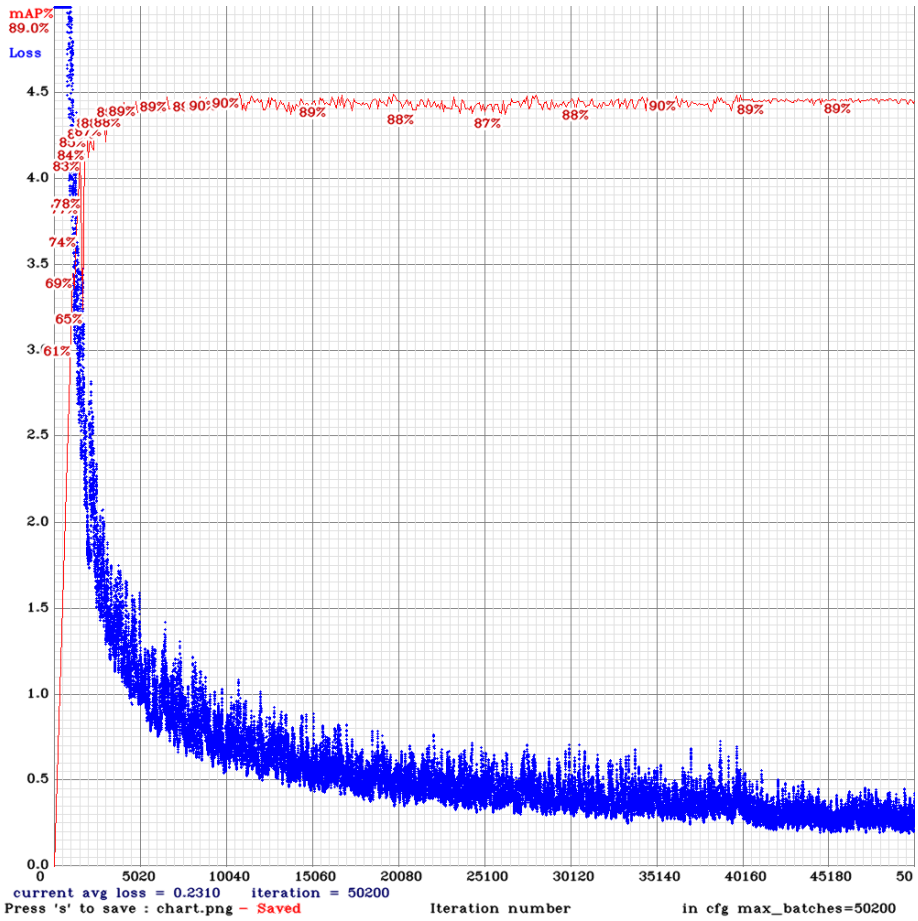
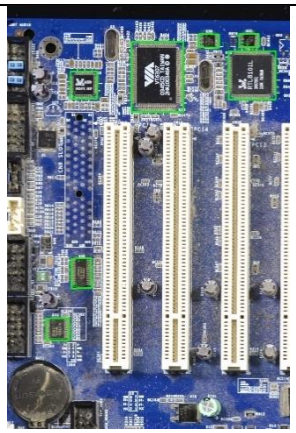
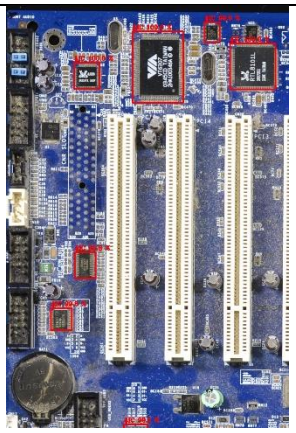
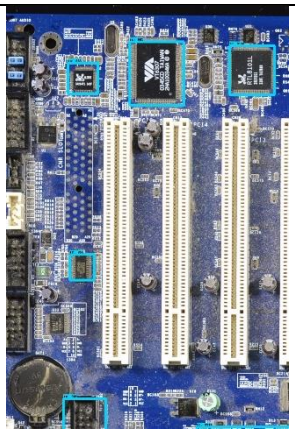
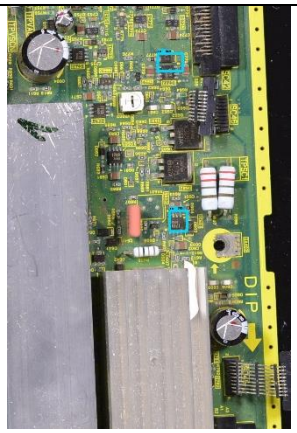
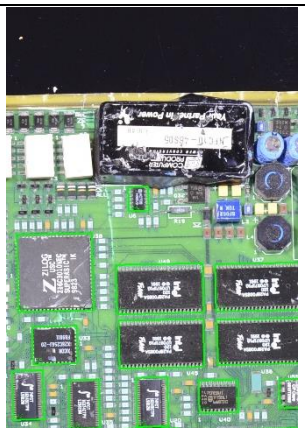
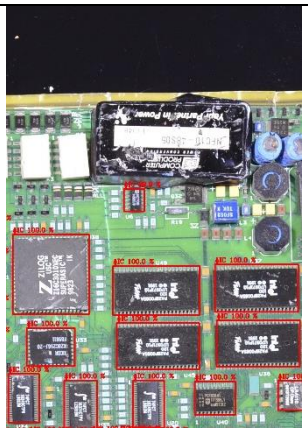
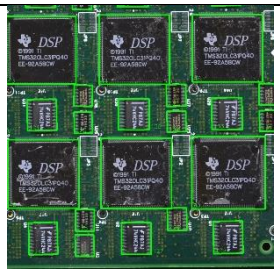
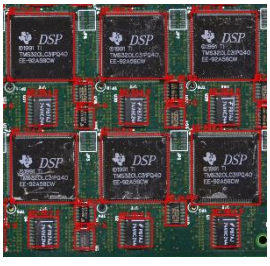
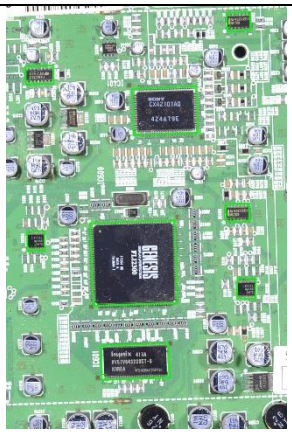
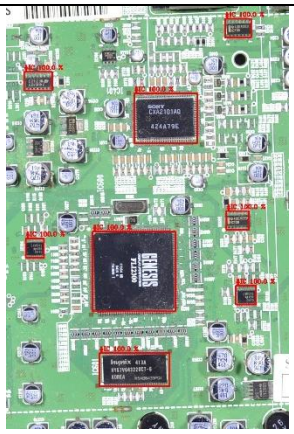
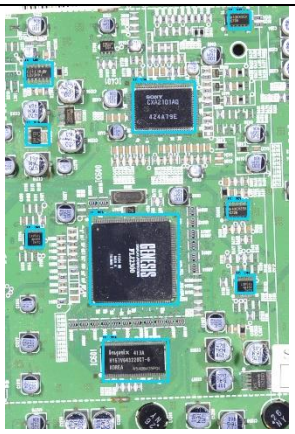
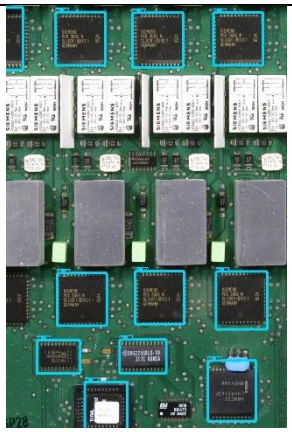


Figure 4.5: YoloV3 Loss and mAP Training Graph

The checkpoint file that produced the highest mAP is used to run inference on six test images that selected from the PCB IC component dataset. Example of inference results (six selected test images) are shown in Table 4.5. We have zoom in to some part of image that has IC components to maximize the viewing experience. As observed, better performance of YoloV3 compared to Faster RCNN (i) the number of false positive is reduced and (ii) the intersection of union (IOU) between ground truth and detected boxes is increased as the depth of the CNN model increased since YoloV3 introduced a new scaling method to detect a small object precisely, using 3 different scales with 27 different size of anchor boxes (3 scales x 9 anchor boxes for each scale) that can be easily fitted on any object size and score a huge improvement on mAP value.

Table 4.5: Ground Truth with YoloV3 and Faster RCNN Detections

Ground Truth	YoloV3 Inference	Faster RCNN Inference
		
7 Ground Truth	6 TP, 1 FN, 0 FP	4 TP, 3 FN, 1 FP
		
5 Ground Truth	5 TP, 0 FN, 0 FP	2 TP, 3 FN, 0 FP
		
12 Ground Truth	12 TP, 0 FN, 0 FP	9 TP, 3 FN, 0 FP
		

21 Ground Truth	21 TP, 0 FN, 0 FP	6 TP, 15 FN, 0 FP
		
8 Ground Truth	8 TP, 0 FN, 0 FP	8 TP, 0 FN, 1 FP
		
14 Ground Truth	14 TP, 0 FN, 0 FP	11 TP, 3 FN, 0 FP

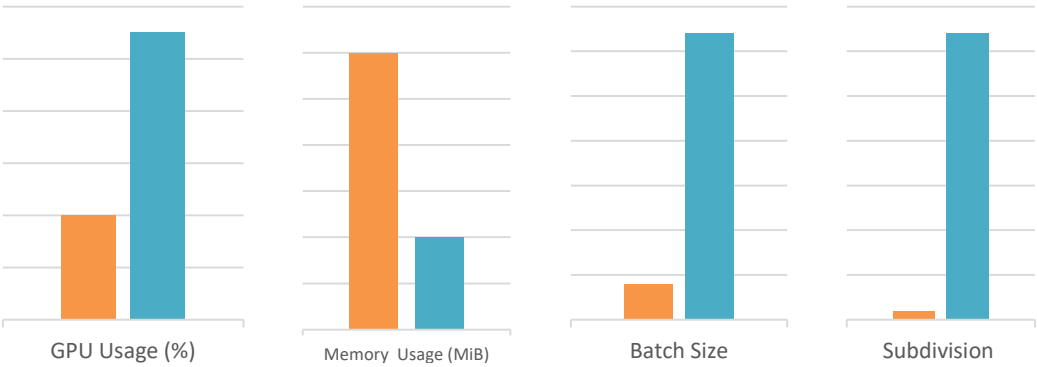
4.3.2 Number of training iterations versus training time and memory usage

The training configuration file of YoloV3 and Faster RCNN has been configured for 50,200 iterations as maximum training iteration both models has been trained using same environment and same hardware capabilities for preforming our comparison fairly. According to Figure 4.6, GPU and memory usage have been measured after 1000 iteration because the GPU needs some warm-up time and does not show an accurate and stable values during first iterations with fixed batch size specified in training's configuration file. We can notice different training batch size between Faster RCNN and YoloV3, where Faster RCNN has been using batch size of 16 and YoloV3 has been using batch size of 128. These

differences due to 2 main reasons, firstly YoloV3 is using 2 parameters (Batch size and subdivision) where Faster RCNN is using batch size parameter only, that makes YoloV3 can load higher batch with lower memory usage over Faster RCNN, secondly the input image size of YoloV3 is 416 x 416 while Faster RCNN is 1024 x 600. According to these two investigations, we are required to decrease the training batch size of Faster RCNN since our GPU memory is 12 GB. Furthermore, a training time of both model has been recorded in Table 4.6, we can notice that YoloV3 has spent almost 4 days more than Faster RCNN to train on 50200 iteration (using same number of iterations) and 1304 images for training and testing dataset (using same number of image dataset), this is due to higher number of convolution layers and complexity of YoloV3 topology compared to Faster RCNN. Finally, Faster RCNN is using Tensorflow framework-based python programming language and YoloV3 is using darknet based C++.

Table 4.6: Models Training Time

IC detection models	Training Time
Faster RCNN	2 days, 4 hours, 33 minutes
Yolo V3	6 days, 14 hours, 41 minutes



	GPU Usage (%)	Memory Usage (MiB)	Batch Size	Subdivision
Faster RCNN	82%	8.6	16	1
YOLOV3	89%	7.8	128	32

Figure 4.6: Faster RCNN and YoloV3 Parameters Comparison

4.3.3 Inference time using different hardware accelerator

In this experiment, the time taken to load an image into model is included when the inference time of each model is measured. For Faster RCNN inference, we have used 2 different inference codes released by TensorFlow object detection API for GPU unoptimized model and a written code for CPU optimized model inference. For YoloV3 inference another 2 different inference codes released by darknet for GPU unoptimized model and a written code for CPU optimized model inference. However, the inference code released by TensorFlow object detection API is not suitable for this experiment as it requires the test dataset conversion into .tfrecord file first before the inference process is started. This is not practical for real time application because conversion to .tfrecord will increase the waiting time. Therefore, the demo code released in TensorFlow object detection API is used to measure the inference time for Faster RCNN using GPU. The demo code released in TensorFlow object detection API used Python Imaging Library (PIL) to load the input images while YoloV3 used OpenCV library to load the input images. To get a fair comparison in measure the inference time, OpenCV 3.4 library is used for both models. By using OpenCV to load the input images, it can speed up the entire inference process by 13 times as compared to PIL. The inference is carried out on image by image loaded by the user. The inference batch size is set to 1 and the test image are resized to 418 x 418 pixels for YoloV3 and 1024 x 600 for Faster RCNN. The inference time for both models using different hardware is shown in Figure 4.7. according to the figure, it is clearly shown that Yolo V3 is always much faster than Faster RCNN, starting by running our inference using NVIDIA GPU 1080Ti with unoptimized model produced by darkent training for YoloV3 and tensorflow object detection API for Faster RCCNN, YoloV3 is able to infer at 15 milliseconds

per frame while Faster RCNN score 2.40 seconds per frame. Meanwhile we successfully were able to infer YoloV3 using CPU only on 9.67 second per frame, while a compilation of Faster RCNN on CPU has been failed. However, to be able to run our system on embedded intel device with acceptable performance, we have used OPENVINO toolkit for optimizing our weight and producing the intermediate representation (IR) of our weight, so we have successfully infer our models on Intel CPU i7 and Pentium using our inspection software to obtain the result of inference time for each model. YoloV3 has inferred each frame on 800 milliseconds while Faster RCNN on 1.14 seconds, we can analysis Faster RCNN optimized model has scored higher FPS using Intel CPU i7 than NVIDIA GPU 1080Ti. Finally using UP^2 board with Pentium Intel CPU, Yolo V3 and Faster RCNN has scored inference rate on 3.88 seconds and 4.92 seconds per frame respectively.

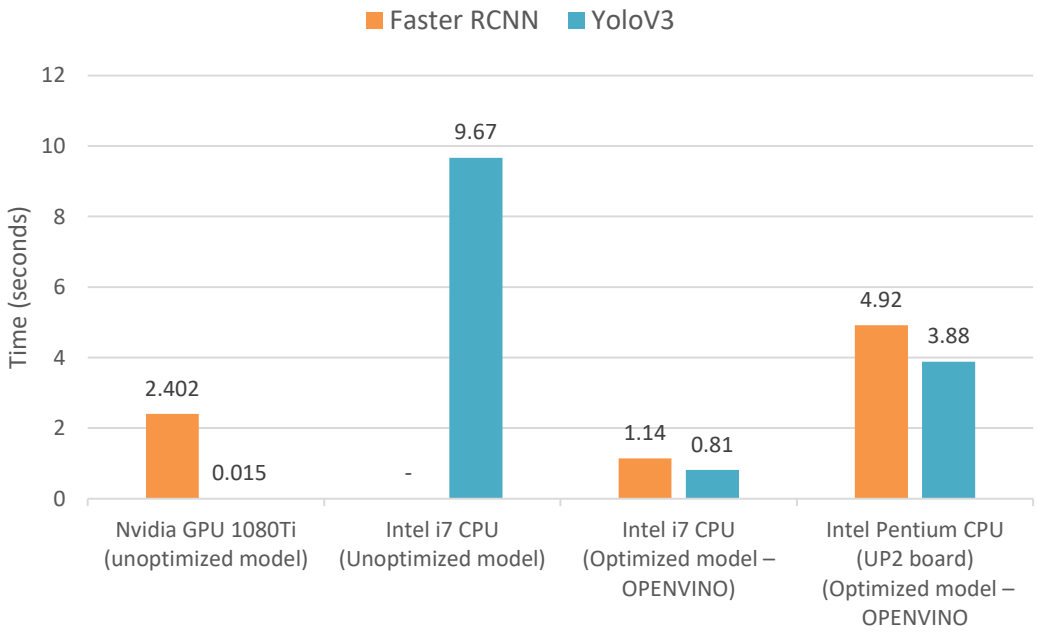


Figure 4.7: Faster RCNN and YoloV3 Frame Inference Time Comparison

4.4 Inspection Software

Sample of our Automated vision inspection based IC component locator software has shown in Figure 4.8, we have loaded one PCB image using (Open Image Button) on right hand side of screen, then we are able to choose which model to infer our image from (Detection Network) dropdown control, we can choose either YoloV3 or Faster RCNN. Then once we are already for inspection process, we press the green button, so a detection result will be displayed on an image view, number of IC detected, and inference time will be displayed at bottom right side of our software. Finally, a detection file (recipe) of our inspection will be saved for each PCB, this file will include all the detection information and precise location of each IC.

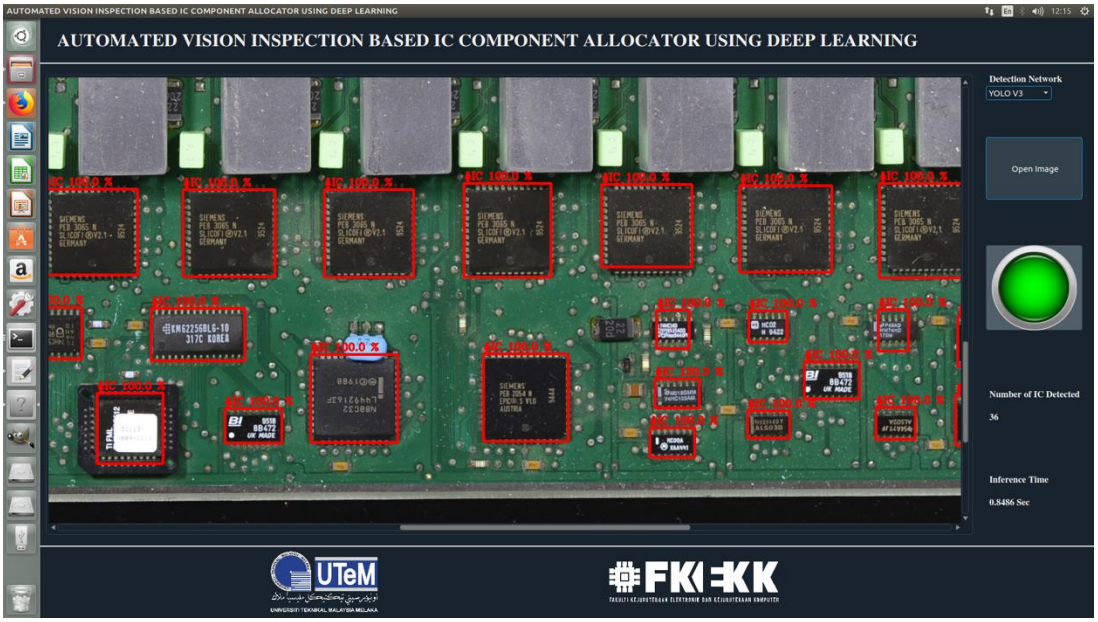


Figure 4.8: Automated Vision Inspection Based IC Component Locator Software

CHAPTER 5

CONCLUSION AND FUTURE WORKS

5.1 Conclusion

Automated vision inspection based IC component locator software is designed and developed using latest object detection deep learning models available. Currently, inspection PCB factories are using operators to draw a bounding box around each component and select component's family, type and other parameters that can help in specifying suitable inspection algorithm. This operation is time costly and needs much efforts and concentration of the operator for each new design of PCB on production line.

An open source PCB dataset has been used to perform our training on different deep learning network (i) YOLOV3 (ii) Faster RCNN. To perform our training, we developed an annotation tools to annotate the PCB dataset that contains 748 images

from a recycling facility, captured under representative conditions using a professional DSLR camera. But, out of the 748 images, only 163 images have been chosen for the project. Using the developed annotation tool, these 163 images have been augmented to become 1304 images total for training and validation using rotation, brightness and noise level augmentation techniques with the help of OpenCV library and python programming language. To start our training of Faster RCNN and YoloV3, a ground truth annotation of our dataset has converted to PASCAL VOC and Yolo format.

In our training experiment we have used a machine with Intel Xeon E5-2620 v3 processor with 47GB RAM. The GPU card used is NVIDIA GeForce GTX-1080 TI. The OS is 64-bits Ubuntu 16.04 LTS with TensorFlow version 1.12.0 precompiled installed via PIP library, Protobuf 3.0.0 and OpenCV version 3.4.0 with CUDA 9.0 and cuDNN v7.0. Both models have trained on same dataset of images with 50200 iterations, then comparison has taken a place according to mAP score, loss value, training time and memory usage. We can conclude that YoloV3 has score higher mAP and lower loss than Faster RCNN, since after 50200 iteration YoloV3 has scored mAP of 0.89 and loss of 0.23, while Faster RCNN has scored mAP of 0.43 and loss of 0.32. we analysis this result and elaborated the reasons of YoloV3 better performance, since YoloV3 introduced a new scaling method to detect a small object preciously, using 3 different scales with 27 different size of anchor boxes (3 scales x 9 anchor boxes for each scale) that can be easily fitted on any object size and score a huge improvement on mAP value. We conclude that training time for both networks was varying due to change of input image size, training batch size and complexity of each networks architecture since YoloV3 has spent 6 days, 14 hours, 41 minutes with batch size of 128 and image size of 418 x 418 while Faster RCNN has spent 2 days, 4 hours, 33 minutes with batch size of 16 and image size of 1024 x 600.

Finally, a model optimization has been applied for both models weight using OPENVINO to be able to accelerate our developed software in any Intel CPU device, so we were able to infer our model on the edge and preforming a comparison in term of inference time per frame. YoloV3 has inferred each frame on 800 milliseconds while Faster RCNN on 1.14 seconds using Intel CPU i7. Furthermore, UP^2 board with Pentium Intel CPU, Yolo V3 and Faster RCNN has scored inference rate on 3.88 seconds and 4.92 seconds per frame respectively.

5.2 Future work recommendation

5.2.1 Collecting more PCB dataset

Since we have used an open source dataset of 163 PCB Images only, a suggestion of collecting more dataset that varies in different key aspect like Camera resolution, environment status and PCB clarity will help in preforming better training in term of mAP and loss scores, meanwhile it will help to generalize our models detection performance.

5.2.2 Comparison over more deep learning models

By train different models that has different architectures and make a comparison over these models in term of mAP and inference time with other different parameters that have been discussed, we shall be able to observe a model that can score higher mAP or find a model that we can rely on its higher inference rate according to our application.

5.2.3 Optimizing using different toolkit for Intelligent on the edge

The TensorRT™ includes a deep learning inference optimizer and runtime that delivers low latency and high throughput for deep learning inference applications. The TensorRT™ provides the optimization as follow:

i. Maximizes throughput by quantizing models to INT8 while preserving accuracy.
ii. Optimizes use of GPU memory and bandwidth by fusing nodes in a kernel.
iii. Selects best data layers and algorithms based on target GPU platform.
iv. Minimizes memory footprint and re-uses memory for tensors efficiently.
v. Scalable design to process multiple input streams in parallel.

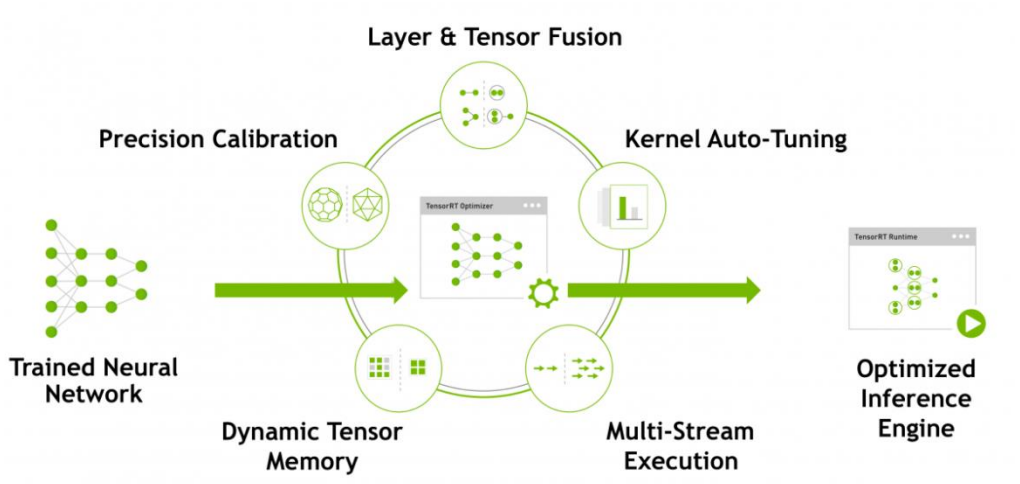


Figure 5.1: Tensor RT Optimization Tool Kit [28]

REFERENCES

[1] David T.Lee,A computerized automatic inspection system for complex printed thick film patterns, SPIEAppl.Electron.Imaging Syst. 143,1978,172-177.

[2] William K.Pratt, Image detection and registration, Digital Image Processing, pp.551-566;WileyInterscience, New York,1978.

[3] Ahmed M.Darwish and Anil K.Jain, A Rule Based Approach for Visual Pattern Inspection, IEEE Transactions on Pattern Analysis and Machine Intelligence.

[4] S.H.Oguz and L.Onural, an automated system for design-rule based visual inspection of printed circuit boards, in Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacremento, CA.

[5] S.Birecik,M.Sezgin, I.O.Bucak, D.Demir,B.Sankur, E.Anarim, Baskili Devre Paketlerindeki Lehim Adacýklarýnýn Matematiksel Morfoloji Yardýmýyla Ýncelenmesi,Sinyal Iþleme ve Uygulamalari.

[6] A.Rosenfeld and A.C.Kak,Digital Picture Processing, Vol. I,Academic Press, Orlando, FL.

[7] Y.Hara, H. Doi , K.Karasaki, T.lida, A system for PCB automated inspection using fluorescent light, IEEE Trans. Pattern Anal.Mach. Intel.

[8] J. Redmon and A. Farhadi, YOLOv3: An Incremental Improvement, arXiv:18 (2018).

[9] S. Ren, K. He, R. Girshick, J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks arXiv:15 (2015)

[10] C. Pramerdorfer and M. Kampel: “A Dataset for Computer-Vision-Based PCB Analysis”, Machine Vision Applications (to appear), 2015.

[11] R. Brunelli, Template Matching Techniques in Computer Vision: Theory and Practice, Wiley,2009

[12] Aksoy, M. S., O. Torkul, and I. H. Cedimoglu. "An industrial visual inspection system that uses inductive learning."

[13] Kyriacou, Theocharis, Guido Bugmann, and Stanislao Lauria. "Vision-based urban navigation procedures for verbally instructed robots." Robotics and Autonomous Systems 51.1 (April 30, 2005)

[14] WANG, CHING YANG, Ph.D. "EDGE DETECTION USING TEMPLATE MATCHING (IMAGE PROCESSING, THRESHOLD LOGIC, ANALYSIS, FILTERS)"

[15] Talmi, Itamar; Mechrez, Roey; Zelnik-Manor, Lihi (2016-12-07). "Template Matching with Deformable Diversity Similarity".

[16] Preprint repository arXiv achieves milestone million uploads". Physics Today. 2014.

[17] Zhang, Richard; Isola, Phillip; Efros, Alexei A.; Shechtman, Eli; Wang, Oliver (2018-01-11). "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric".

[18] Talmi, Mechrez, Zelnik-Manor (2016). "Template Matching with Deformable Diversity Similarity".

[19] Li, Yuhai, L. Jian, T. Jinwen, X. Honbo. “A fast-rotated template matching based on point feature.”

[20] B. Sirmacek, C. Unsalan. “Urban Area and Building Detection Using SIFT Keypoints and Graph Theory”.

[21] Dalal, Navneet (2005). "Histograms of oriented gradients for human detection”.

[22] Liu, Wei (October 2016). SSD: Single shot multibox detector. European Conference on Computer Vision. Lecture Notes in Computer Science.

[23] A Practical Introduction to Computer Vision with OpenCV by Kenneth Dawson-Howe © Wiley & Sons Inc. 2014.

[24] Li, Yang, Feng, Zhou, Chakradhar, “Optimizing Memory Efficiency for Deep Convolutional Neural Networks on GPUs”, North Carolina State University, Department of Electrical and Computer Eng, October 12, 2016.

[25] Schlegel, Daniel. “Deep Machine Learning on GPUs”, University of Heidelberg, January 28, 2015.

[26] Filtered PCB Dataset, https://drive.google.com/open?id=185EkUYi6r-wd4g3KnGneY3QcC_l_vZFj3

[27] OpenVINO™ toolkit Documentation: https://software.intel.com/en-us/OpenVINO-toolkit/documentation/featured

[28] TensorRT™ Toolkit Developer Guide: https://docs.nvidia.com/deeplearning-/sdk/tensorrt-developer-guide/index.html

APPENDICES

Appendix A

UTeM Annotation and Augmentation Tool

```
'''
Annotate a PCB image.
Ibrahim Soliman Mohamed, 4BENG, FKEKK, UTeM, Malaysia
'''
import glob
import cv2
import numpy as np
from PyQt4 import QtGui, QtCore
import sys
import os.path

class Annot:

    def __init__(self, rect, text=''):
        if rect is None:
            rect = ((0, 0), (0, 0), 0)

        self.rect = rect
        self.text = text

    def __repr__(self):
        return 'Annotation {} "{}"'.format(self.rect, self.text)


def parse_annotation_file(path):
    lines = None
    with open(path) as f:
        lines = [l.strip().split() for l in f.readlines()]

    ret = []
    for l in lines:
        l = [x.strip() for x in l]
        if len(l) < 5:
            raise Exception('Failed to parse line "{}"'.format(l))

        rect = [float(s) for s in l[:5]]
        text = '' if len(l) == 5 else ' '.join(l[5:])
```

```
        ret.append(Annot((tuple(rect[0:2]),   tuple(rect[2:4]),   rect[4]),
text))

    return ret


def write_annotation_file(data, to):
    with open(to, 'w') as f:
        for d in data:
            f.write('{:.0f}      {:.0f}      {:.0f}      {:.0f}      {:.3f}
{}\n'.format(d.rect[0][0],   d.rect[0][1],   d.rect[1][0],   d.rect[1][1],
d.rect[2], d.text))

def write_annotation_file_90(data, to,h):
    with open(to, 'w') as f:
        for d in data:
            f.write('{:.0f}  {:.0f}  {:.0f}  {:.0f}  {:.3f}  {}\n'.format((h-
d.rect[0][1]),  d.rect[0][0],  d.rect[1][0],d.rect[1][1],  (d.rect[2]+90.0),
d.text))

def write_annotation_file_180(data, to,h,w):
    with open(to, 'w') as f:
        for d in data:
            f.write('{:.0f}  {:.0f}  {:.0f}  {:.0f}  {:.3f}  {}\n'.format((w-
d.rect[0][0]),  (h-d.rect[0][1]),  d.rect[1][0],  d.rect[1][1],  d.rect[2],
d.text))

def write_annotation_file_270(data, to,w):
    with open(to, 'w') as f:
        for d in data:
            f.write('{:.0f}      {:.0f}      {:.0f}      {:.0f}      {:.3f}
{}\n'.format((d.rect[0][1]), (w-d.rect[0][0]), d.rect[1][0], d.rect[1][1],
(d.rect[2]+90.0), d.text))

def write_annotation_file_flip(data, to,w):
    with open(to, 'w') as f:
        for d in data:
            f.write('{:.0f}  {:.0f}  {:.0f}  {:.0f}  {:.3f}  {}\n'.format((w-
d.rect[0][0]),   d.rect[0][1],   d.rect[1][0],   d.rect[1][1],   d.rect[2],
d.text))


def mat_to_qimage(mat):
    tmp = cv2.cvtColor(mat, cv2.COLOR_BGR2RGB)
    h, w, bpc = tmp.shape
    return QtGui.QImage(tmp.data, w, h, bpc*w, QtGui.QImage.Format_RGB888)

def increase_brightness(img, value=30):
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(hsv)

    lim = 255 - value
    v[v > lim] = 255
    v[v <= lim] += value

    final_hsv = cv2.merge((h, s, v))
    img = cv2.cvtColor(final_hsv, cv2.COLOR_HSV2BGR)
    return img

class MouseLabel(QtGui.QLabel):
```

```
    def __init(self, parent):
        QtGui.QLabel.__init__(self, parent)

    def mouseReleaseEvent(self, ev):
        self.emit(QtCore.SIGNAL('clicked(int, int, int)'), ev.x(), ev.y(),
ev.button())


class Window(QtGui.QWidget):

    def __init__(self):


        QtGui.QWidget.__init__(self)
        self.setWindowTitle('UTeM PCB Annotator')
        self.setGeometry(100, 100, 1200, 1000)
        screen = QtGui.QDesktopWidget().screenGeometry()
        self.image_path = None
        self.buttonopenFolder = QtGui.QToolButton(self)
        self.buttonopenFolder.setText('   Open Folder   ')
        self.buttonopenFolder.setIcon(QtGui.QIcon('./res/image.png'))
        self.buttonopenFolder.setIconSize(QtCore.QSize(80,80))
        self.buttonopenFolder.setCheckable(True)
        self.buttonopenFolder.setFixedSize(130,110)

self.buttonopenFolder.setToolButtonStyle(QtCore.Qt.ToolButtonTextUnderIcon
)
        self.buttonopenFolder.move(1710,15)
        self.buttonopenFolder.clicked.connect(self.openFloder)

        self.nextImageButton = QtGui.QToolButton(self)
        self.nextImageButton.setText('Previous \nImage')
        self.nextImageButton.setIcon(QtGui.QIcon('./res/left.png'))
        self.nextImageButton.setIconSize(QtCore.QSize(35,35))
        self.nextImageButton.setFixedSize(60,90)
        self.nextImageButton.setCheckable(True)

self.nextImageButton.setToolButtonStyle(QtCore.Qt.ToolButtonTextUnderIcon)
        self.nextImageButton.move(1710,135)
        self.nextImageButton.clicked.connect(self.fetchPreviousImage)

        self.previousImageButton = QtGui.QToolButton(self)
        self.previousImageButton.setText('Next \nImage')
        self.previousImageButton.setIcon(QtGui.QIcon('./res/rightF.png'))
        self.previousImageButton.setIconSize(QtCore.QSize(35,35))
        self.previousImageButton.setFixedSize(60,90)
        self.previousImageButton.setCheckable(True)

self.previousImageButton.setToolButtonStyle(QtCore.Qt.ToolButtonTextUnderI
con)
        self.previousImageButton.move(1775,135)
        self.previousImageButton.clicked.connect(self.fetchNextImage)

        self.saveYoloFormat = QtGui.QToolButton(self)
        self.saveYoloFormat.setText('   Save Annot    ')
        self.saveYoloFormat.setIcon(QtGui.QIcon('./res/save.png'))
        self.saveYoloFormat.setIconSize(QtCore.QSize(80,80))
        self.saveYoloFormat.setCheckable(True)
        self.saveYoloFormat.setFixedSize(130,120)

self.saveYoloFormat.setToolButtonStyle(QtCore.Qt.ToolButtonTextUnderIcon)
```

```
        self.saveYoloFormat.move(1710,250)
        self.saveYoloFormat.clicked.connect(self.saveAnnotation)

        l1 = QtGui.QLabel(self)
        l1.setText("Data Augmentation")
        l1.move(1700,400)

        utemlogo = QtGui.QLabel(self)
        pixmap = QtGui.QPixmap('./res/utem.png')

utemlogo.setPixmap(pixmap.scaled(150,100,QtCore.Qt.KeepAspectRatio))
        utemlogo.move(1700,890)
        fkekklogo = QtGui.QLabel(self)
        pixmap = QtGui.QPixmap('./res/fkekk.png')

fkekklogo.setPixmap(pixmap.scaled(150,100,QtCore.Qt.KeepAspectRatio))
        fkekklogo.move(1700,1000)


        self.rotateImage = QtGui.QToolButton(self)
        self.rotateImage.setText(' Roation ')
        self.rotateImage.setIcon(QtGui.QIcon('./res/rotate.png'))
        self.rotateImage.setIconSize(QtCore.QSize(70,70))
        self.rotateImage.setCheckable(True)
        self.rotateImage.setFixedSize(130,95)

self.rotateImage.setToolButtonStyle(QtCore.Qt.ToolButtonTextUnderIcon)
        self.rotateImage.move(1710,430)
        self.rotateImage.clicked.connect(self.rotate3Image)

        self.flipping = QtGui.QToolButton(self)
        self.flipping.setText('  Flipping ')
        self.flipping.setIcon(QtGui.QIcon('./res/FLIP_F.png'))
        self.flipping.setIconSize(QtCore.QSize(70,70))
        self.flipping.setCheckable(True)
        self.flipping.setFixedSize(130,95)

self.flipping.setToolButtonStyle(QtCore.Qt.ToolButtonTextUnderIcon)
        self.flipping.move(1710,530)
        self.flipping.clicked.connect(self.flippingImage)

        self.addnoise = QtGui.QToolButton(self)
        self.addnoise.setText('   Salt & Papper Noise   ')
        self.addnoise.setIcon(QtGui.QIcon('./res/noise.png'))
        self.addnoise.setIconSize(QtCore.QSize(70,70))
        self.addnoise.setCheckable(True)
        self.addnoise.setFixedSize(130,95)

self.addnoise.setToolButtonStyle(QtCore.Qt.ToolButtonTextUnderIcon)
        self.addnoise.move(1710,630)
        self.addnoise.clicked.connect(self.addnoiseImage)

        self.addBright = QtGui.QToolButton(self)
        self.addBright.setText('   Brightness   ')
        self.addBright.setIcon(QtGui.QIcon('./res/bright.png'))
        self.addBright.setIconSize(QtCore.QSize(70,70))
        self.addBright.setCheckable(True)
        self.addBright.setFixedSize(130,95)

self.addBright.setToolButtonStyle(QtCore.Qt.ToolButtonTextUnderIcon)
        self.addBright.move(1710,730)
```

```
        self.addBright.clicked.connect(self.addBrightImage)



        self.label_image = MouseLabel(self)
        self.scroll = QtGui.QScrollArea(self)
        self.scroll.setWidget(self.label_image)

        self.scroll.setGeometry(0,          0,          (0.88*screen.width()),
(0.98*screen.height()))
        self.showMaximized()

        self.connect(self.label_image,   QtCore.SIGNAL('clicked(int,   int,
int)'), self.point_selected)
    def openFloder(self):
        dlg = QtGui.QFileDialog(self)
        dlg.setFileMode(QtGui.QFileDialog.AnyFile)
        dlg.setFilter("Image files (*.jpg)")
        dlg.show()
        if dlg.exec_():
            filenames = str(dlg.selectedFiles()[0])
            img = cv2.imread(filenames)
            annot_name      =      os.path.join(os.path.dirname(filenames),
'{}.txt'.format(os.path.splitext(filenames)[0]))
            annot_data = []
            if os.path.isfile(annot_name):
                print('Loading existing annotation file')
                annot_data = parse_annotation_file(annot_name)
            self.newImage(filenames, img, annot_name, annot_data)
            self.redraw()
            #print(filenames)
    def newImage(self, image_path, image, annot_path, annot):
        self.image_path = image_path
        self.image = image.copy()
        self.annot_path = annot_path
        self.annot = annot
        self.annot_coords = [[0, 0], [0, 0], [0, 0], [0, 0]]  # x;y
        self.annot_idx = 0

    def point_selected(self, x, y, button):
        print(str(button) + '  button ')
        if button != 1 and button != 2:
            return

        if button == 1:
            # new annotation point

            self.annot_coords[self.annot_idx][0] = x
            self.annot_coords[self.annot_idx][1] = y

            self.annot_idx += 1
            if self.annot_idx == 4:
                rect = cv2.minAreaRect(np.array(self.annot_coords))
                self.annot.append(Annot(rect))

                self.annot_idx = 0

        if button == 2:
            # remove existing annotation

            id = 0
```

```
            del_id = -1
            for an in self.annot:
                tmp = np.zeros(self.image.shape[:2], dtype=np.uint8)
                bp = cv2.cv.BoxPoints(an.rect)
                bp = np.int0(bp)
                cv2.drawContours(tmp, [bp], 0, (255), -1)

                if tmp[y, x] > 0:
                    del_id = id
                    break

                id += 1

            if del_id >= 0:
                del self.annot[del_id]

        self.redraw()

    def redraw(self):
        vis = self.image.copy()
        id = 1
        for an in self.annot:
            bp = cv2.boxPoints(an.rect)
            bp = np.int0(bp)
            cv2.drawContours(vis, [bp], 0, (0, 255, 0),3)

            cv2.putText(vis, '{}'.format(id), (bp[0, 0]+5, bp[0, 1]-5),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0))
            id += 1

        for idx in np.arange(self.annot_idx):
            cv2.line(vis,                    tuple(self.annot_coords[idx]),
tuple(self.annot_coords[idx]), (0, 0, 255), 2)


self.label_image.setPixmap(QtGui.QPixmap.fromImage(mat_to_qimage(vis)))
        self.label_image.setGeometry(QtCore.QRect(0,    0,    vis.shape[1],
vis.shape[0]))
        cv2.imwrite('/home/ibrahim/hi.jpg',vis )

    def fetchNextImage(self):
        if self.image_path is not None:
            path, _ = os.path.split(self.image_path)
            filesImage = glob.glob(path+'/*.jpg')
            indexOfImage = filesImage.index(self.image_path)
            if len(filesImage) > indexOfImage:
                img = cv2.imread(filesImage[indexOfImage+1])
                annot_name                                                    =
os.path.join(os.path.dirname(filesImage[indexOfImage+1]),
'{}.txt'.format(os.path.splitext(filesImage[indexOfImage+1])[0]))
                annot_data = []
                if os.path.isfile(annot_name):
                    print('Loading existing annotation file')
                    annot_data = parse_annotation_file(annot_name)
                self.newImage(filesImage[indexOfImage+1], img, annot_name,
annot_data)
                self.redraw()
                print("path "+str(indexOfImage))
    def fetchPreviousImage(self):
        if self.image_path is not None:
            path, _ = os.path.split(self.image_path)
```

```
            filesImage = glob.glob(path+'/*.jpg')
            indexOfImage = filesImage.index(self.image_path)
            if indexOfImage > 0:
                img = cv2.imread(filesImage[indexOfImage-1])
                annot_name                                                    =
os.path.join(os.path.dirname(filesImage[indexOfImage-1]),
'{}.txt'.format(os.path.splitext(filesImage[indexOfImage-1])[0]))
                annot_data = []
                if os.path.isfile(annot_name):
                    print('Loading existing annotation file')
                    annot_data = parse_annotation_file(annot_name)
                self.newImage(filesImage[indexOfImage-1], img, annot_name,
annot_data)
                self.redraw()
                print("path "+str(indexOfImage))
    def saveAnnotation(self):
        print('Writing annotation data to "{}"'.format(self.annot_path))
        write_annotation_file(self.annot, self.annot_path)

    def rotate3Image(self):
        rotated1 = cv2.rotate(self.image, cv2.ROTATE_90_CLOCKWISE)
        write_annotation_file_90(self.annot,
(os.path.splitext(self.annot_path)[0]+'r90.txt'),self.image.shape[0])

cv2.imwrite((os.path.splitext(self.annot_path)[0]+'r90.jpg'),rotated1)
        rotated2 = cv2.rotate(self.image, cv2.ROTATE_180)
        write_annotation_file_180(self.annot,
(os.path.splitext(self.annot_path)[0]+'r180.txt'),self.image.shape[0],self
.image.shape[1])

cv2.imwrite((os.path.splitext(self.annot_path)[0]+'r180.jpg'),rotated2)
        rotated3 = cv2.rotate(self.image, cv2.ROTATE_90_COUNTERCLOCKWISE)
        write_annotation_file_270(self.annot,
(os.path.splitext(self.annot_path)[0]+'r270.txt'),self.image.shape[1])

cv2.imwrite((os.path.splitext(self.annot_path)[0]+'r270.jpg'),rotated3)

    def addnoiseImage(self):
        img = self.image.copy()
        noiseImage = cv2.randn(img,(0,0,0),(5,5,5))
        write_annotation_file(self.annot,
(os.path.splitext(self.annot_path)[0]+'n1.txt'))

cv2.imwrite((os.path.splitext(self.annot_path)[0]+'n1.jpg'),(self.image+no
iseImage))
        img2 = self.image.copy()
        noiseImage2 = cv2.randn(img2,(0,0,0),(10,10,10))
        write_annotation_file(self.annot,
(os.path.splitext(self.annot_path)[0]+'n2.txt'))

cv2.imwrite((os.path.splitext(self.annot_path)[0]+'n2.jpg'),(self.image+no
iseImage2))

    def addBrightImage(self):
        frameBright1 = increase_brightness(self.image, value=10)
        write_annotation_file(self.annot,
(os.path.splitext(self.annot_path)[0]+'b1.txt'))

cv2.imwrite((os.path.splitext(self.annot_path)[0]+'b1.jpg'),(frameBright1)
)
        frameBright2 = increase_brightness(self.image, value=20)
```

```
        write_annotation_file(self.annot,
(os.path.splitext(self.annot_path)[0]+'b2.txt'))

cv2.imwrite((os.path.splitext(self.annot_path)[0]+'b2.jpg'),(frameBright2)
)

    def flippingImage(self):
        horizontal_img = cv2.flip( self.image, 1 )
        write_annotation_file_flip(self.annot,
(os.path.splitext(self.annot_path)[0]+'f.txt'),self.image.shape[1])

cv2.imwrite((os.path.splitext(self.annot_path)[0]+'f.jpg'),(horizontal_img
))
    def closeEvent(self, event):
        reply = QtGui.QMessageBox.question(self, 'Save?', 'Save changes to
file?', QtGui.QMessageBox.Yes | QtGui.QMessageBox.No, QtGui.QMessageBox.No)
        if reply == QtGui.QMessageBox.Yes:
            print('Writing          annotation          data          to
"{}"'.format(self.annot_path))
            write_annotation_file(self.annot, self.annot_path)

        event.accept()


app = QtGui.QApplication(sys.argv)
win = Window()
win.show()

sys.exit(app.exec_())
```

Appendix B

Converting from original annotation format to Yolo annotation format

```
import glob
import cv2
import numpy as np
import sys
import os.path

class Annot

    def __init__(self, rect, text=''):
        if rect is None:
            rect = ((0, 0), (0, 0), 0)

        self.rect = rect
        self.text = text

    def __repr__(self):
        return 'Annotation {} "{}"'.format(self.rect, self.text)

def write_annotation_file(data, to , h, w):
    with open(to, 'w') as f:
        for d in data:
            f.write('0 {:.6f} {:.6f} {:.6f}
{:.6f}\n'.format((d.rect[0][0]/w), (d.rect[0][1]/h), (d.rect[1][0]/w),
(d.rect[1][1])/h))

def parse_annotation_file(path):
    lines = None
    with open(path) as f:
        lines = [l.strip().split() for l in f.readlines()]

    ret = []
    for l in lines:
        l = [x.strip() for x in l]
        if len(l) < 5:
            raise Exception('Failed to parse line "{}"'.format(l))

        rect = [float(s) for s in l[:5]]
        text = '' if len(l) == 5 else ' '.join(l[5:])
        ret.append(Annot((tuple(rect[0:2]), tuple(rect[2:4]), rect[4]),
text))

    return ret

filestext =
glob.glob('/home/ibrahim/projects/FYP/Dataset/mva15/dataset/*.txt')

for d in filestext:
    annot_name = d
    imagePath = os.path.join(os.path.dirname(d),
'{}.jpg'.format(os.path.splitext(d)[0]))
```

```
    img= cv2.imread(imagePath)
    annot_data = []
    if os.path.isfile(annot_name):
        print('Loading existing annotation file  ' + str(annot_name))
        annot_data = parse_annotation_file(annot_name)
        head, tail = os.path.split(d)
        write_annotation_file(annot_data,
('/home/ibrahim/projects/FYP/Dataset/mva15/Final_Dataset_Yolo/'+tail),img.
shape[0],img.shape[1])
```

Appendix C

Generating Training and Validation text files for Yolo Training

```
import glob, os

# Current directory
current_dir = os.path.dirname(os.path.abspath(__file__))

print(current_dir)

current_dir = '/home/ubuntu/TF_API/FYP/Final_Dataset_Yolo'

# Percentage of images to be used for the valid set
percentage_test = 20;

# Create and/or truncate train.txt and valid.txt
file_train = open('train.txt', 'w')
file_test = open('valid.txt', 'w')

# Populate train.txt and valid.txt
counter = 1
index_test = round(100 / percentage_test)
for pathAndFilename in glob.iglob(os.path.join(current_dir, "*.jpg")):
    title, ext = os.path.splitext(os.path.basename(pathAndFilename))
    if counter == index_test:
        counter = 1
        file_test.write(current_dir + "/" + title + '.jpg' + "\n")
    else:
        file_train.write(current_dir + "/" + title + '.jpg' + "\n")
        counter = counter + 1
```