DEVELOPMENT OF ARTIFICIAL INTELLIGENCE (AI) FOR IMAGE PROCESSING AND CONCEPTUAL DISTANCE COMPUTATION FROM CAMERA FOR PICK AND PLACE OF OIL PALM FRESH FRUIT BUNCH (FFB)

by

SITI ADAWIYYAH SIDDIQA BINTI SAIFUDDIN

(Matrix no: 131208)

Supervisor:

Dr. Khairuddin Mohamed

May 2019

This dissertation is submitted to

Universiti Sains Malaysia

As partial fulfillment of the requirement to graduate with honors degree in BACHELOR OF ENGINEERING (MANUFACTURING ENGINEERING WITH MANAGEMENT)

School of Mechanical Engineering

Engineering Campus

Universiti Sains Malaysia

DECLARATION

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed_______________________________________ (SITI ADAWIYYAH SIDDIQA BINTI SAIFUDDIN)

Date ____

Statement 1

This thesis is the result of my own investigation, except where otherwise stated. Other sources are acknowledged by giving explicit references. Bibliography/references are appended.

Signed_______________________________________ (SITI ADAWIYYAH SIDDIQA BINTI SAIFUDDIN)

Date ____

Statement 2

I hereby give consent for my thesis, if accepted, to be available for photocopying and for interlibrary loan, and for title and summary to be made available outside organizations.

Signed_______________________________________ (SITI ADAWIYYAH SIDDIQA BINTI SAIFUDDIN)

Date ____

ACKNOWLEDGEMENT

Foremost, I want to offer this endeavour to The God Almighty for the wisdom He bestowed upon me, the strength, peace of my mind and good health in order to finish this project.

Firstly, I would like to take this opportunity to express my gratitude to my supervisor, Dr. Khairudin Mohamed for his patience, enthusiasm, insightful comments, invaluable suggestions, helpful information, practical advice and unceasing ideas which have helped me tremendously at all times in my research and writing of this thesis.

Next, I am also thankful to Dr. Zahurin Samad for his invaluable guidance, continuous encouragement and constant support in making this research possible while my supervisor is not available in USM for the first semester. They are great professors and lecturers to have as an advisor and mentor throughout my education at Universiti Sains Malaysia (USM).

I would also like to thank Assoc. Prof. Dr. Jamaluddin Abdullah, Dean of School of Mechanical Engineering, Universiti Sains Malaysia for giving me the opportunity to carry out this meaningful project. I would like to express my sincere gratitude to several individuals for supporting me throughout my final year project especially Mr. Hadi for giving me technical support along the journey. He was very accommodating and generous with helping me when I was stucked in problems.

Last but not least, I would like to express my gratitude towards my friends, family and most importantly both of my parents for their kind cooperation and encouragement which give me motivation in the completion of this project. I have no words to express my thanks, but my heart is full of favors received from every person.

TABLE OF CONTENT

LIST OF FIGURE

LIST OF TABLE

ABSTRAK

Di Malaysia, aktiviti pertanian merupakan sektor yang penting kepada manusia terutamanya terhadap industri makanan. Setiap hari, para penyelidik telah melakukan penyelidikan dalam pertanian demi peningkatan kualiti, produktiviti serta mengehadkan kadar kesalahan manusia. Malaysia dikategorikan sebagai pengeluar dan pengeksport kelapa sawit terbesar di dunia. Untuk meningkatkan produktiviti buah kelapa sawit, pelbagai teknik penglihatan mesin dan sistem mekanisasi boleh diaplikasikan. Hal ini adalah untuk mengurangkan penggunaan buruh manusia mengutip hasil tanaman dengan hanya menggunakan sabit. Masalah kekurangan buruh dapat dikurangkan kerana kebanyakan pekerja tidak sanggup melakukan kerja keras tersebut dengan bayaran gaji yang tidak berpatutan. Oleh itu, idea projek ini adalah untuk mewujudkan satu sistem untuk sebuah mesin memilih dan meletakkan hasil tanaman menggunakan bot automatik yang dipasang di trak dengan bantuan kamera langsung. Sebagai langkah awal, satu sistem pengesanan kelapa sawit dicipta menggunakan dua algoritma pemprosesan imej, YOLO dan Tensorflow. Prestasi kedua-dua algoritma ini kemudian ditentukan dari segi tahap keyakinan dan kelajuan untuk memproses imej menggunakan tiga parameter yang berbeza. Kemudian, satu kajian bagaimana kamera mengetahui jarak kelapa sawit yang dikesan dari kamera juga dilakukan dalam projek ini. Sepanjang projek ini, perkara yang dapat disimpulkan ialah algoritma YOLO menyediakan kelajuan yang lebih tinggi sementara Tensorflow mempunyai ketepatan yang lebih baik dalam mengesan kelapa sawit. Konsep untuk mengira jarak antara kamera dan buah kelapa sawit kemudian dicadangkan dalam kajian ini untuk aplikasi dan eksperimen pada masa hadapan.

ABSTRACT

In Malaysia, agriculture activities are the major sector that provides importance to the entire human beings especially in the food industry. Most of the researchers have done research in agriculture every day to aim for development in quality, productivity and limiting probability of human error. Malaysia is categorized as the largest producer and exporter of oil palm globally. In order to increase the productivity of palm oil fruits, various machine-vision techniques and mechanization systems can be applied. This is to reduce the human labor of picking the crops by simply using sickles. The labor shortage problems can be minimized as not many labors are willing to do the hard job while the salary is not reasonable. Therefore, the idea of this project is to create a system for a machine to pick and place the crops using an automated bot mounted at the truck with the aid of live feed camera. As early steps, a system of oil palm FFB detection is being created using two algorithms of image processing, YOLO and Tensorflow Object Detection API. The performance of both algorithms are then determined in terms of confidence level and speed to process the image in three different parameters. Then, a study of how the camera computes the distance of detected oil palm FFB from the camera is done in this project. Throughout this project, it can be deduced that YOLO provides higher speed while Tensorflow performs better accuracy in detecting the oil palm FFB. A concept to compute the distance between camera and the respective oil palm fruit is proposed in this research for future work application and experiment.

CHAPTER I INTRODUCTION

At present, automation of agricultural operations seems to be in demand in order to improve productivity with the help of tools and technology. In recent years, the development of autonomous vehicles in agriculture has become very important significantly. Malaysia is facing a labour shortage in oil palm plantations, estimated to account for 46% of the total industrial workforce. Initiatives are being made to increase the productivity of workers by utilising a wide range of intensive mechanisation technologies[1]. Researchers started to develop more rational and adaptable vehicles for agricultural operations[2]. For example, robots performing agriculture operations such as pick and place of crops by applying the Artificial Intelligence (AI) on them. With the presence of this system, the robotic system will be able to act and react by itself [3].

AI also known as a neural network is an electronic model of the human brain resemblance that is made up of interconnected simple processors. It approaches image processing and pattern recognition that are comprehended as alternatives or improvements from the traditional statistically-based procedures[3]. This system can be implemented in the image processing of the crops in agriculture such as the oil palm fruit to be comprehended by robots for agriculture operations.

The development of AI for image processing is studied by programming the image detection of the oil palm fresh fruit bunch. This image detection can be developed by using a network that predicts a single bounding box and confidence score for each oil palm fruit category in the image. The model captures the whole-image context around the oil palm fruit. However, it cannot handle multiple instances of the same object in the image without replicating the number of outputs for each instance [4].

Next, robotics plays an important role in agricultural production and management. The demand for autonomous and time-saving technology in agriculture is to have efficient farm management[5]. In this experiment too, motion control of the pick and place of oil palm fresh fruit bunch (FFB) are studied which is the distance from the camera to the detected oil palm. In automated control system, motion control can provide advanced machine functionality. It provides the functionality to move the machine tooling or the part itself in a controlled, precision, rotary or linear manner[7]. Figure 1 shows the pick and place motion framework of a robot which starts with input, image processing, motion planning, robot control and ended with the operations of pick and place of the robots using sensors and actuators.

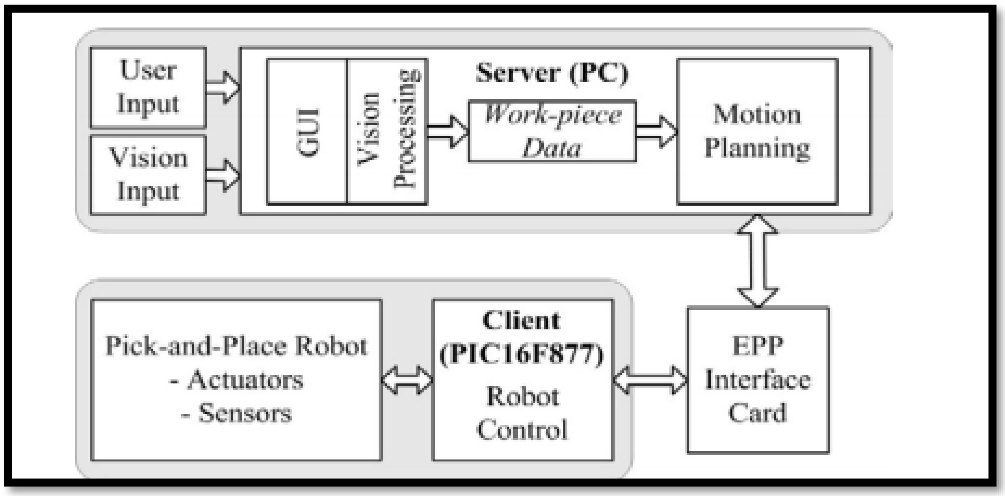


Figure 1: Framework of pick and place motion[8]

1.1 OVERALL STRUCTURE OF THE PROJECT

In this project, two types of framework of neural network are going to be tested under the development of artificial intelligence in image processing system. These two frameworks are named YOLO and Tensorflow Object Detection API. In YOLO, Tensorflow are also being used for pretrained Object Detection models. So, Tensorflow is a part of the models while YOLO is one such model for object detection. The performance of both frameworks are going to be determined with same training set of data which includes 700 mixture of oil palm fruit images captured with real life images as well as Google images. By having the same training data set, the confidence level of the samples for testing purpose can be obtained under different type of conditions (non-scientific). The confidence level of every testing image will vary under different light intensity, environment and appearance of the oil palm FFB.

The Tensorflow framework will be installed in a microprocessor, Raspberry Pi B+ using Python script. The performance of the Tensorflow will be tested to see whether the objects can be detected in a real-time camera which is the Pi Camera mounted in RaspberryPi. However, YOLO framework is still in discussion of machine learning team worldwide to be implemented in microprocessor as the power is not efficient to support the fast algorithms.

A research will be done on how a ‘Sawit bot’ will find the distance of the oil palm fruit and use the claw to pick and place the oil palm FFB on the truck. An ultrasonic sensor will be used because the same concept of car reverse parking sensor will be used. The sensor will detect the location of the oil palm fruit and the camera will start executing the object detection before the ‘Sawit bot’ apply the pick and place operations. A ‘Sawit bot’ is basically a robot to pick and place the oil palm fruit with a claw mounted on the truck with sensor and camera attached to it.

1.2 PROBLEM STATEMENT

In the recent years, crops of oil palm FFB are picked and placed onto the truck manually which involves labor activities. However, labor shortage is the crisis of the recent agriculture production specifically the oil palm FFB industries. Therefore, machine-vision techniques and mechanization systems can be applied as a solution to this problem. For the first step, an image processing system is important to detect the oil palm FFB before the mechanization takes place. Different algorithms of image processing frameworks provide different performance. Besides that, computation of the distance from the camera to the respective oil palm fruit plays an important role in this mechanization of pick and place for the bot to precisely do its operations. In this project, two different algorithms are being compared for the oil palm FFB detection as well as do a research on the concept to find the distance of the camera from the respective oil palm FFB in real situations.

1.3 OBJECTIVES

1. To create a system using open-source softwares for object detection of oil palm FFB using YOLO and Tensorflow Object Detection API algorithms.
2. To determine the performance of YOLO and Tensorflow Object Detection API algorithms using test images of oil palm FFB in three different parameters.
3. To study the concept of distance computation of oil palm FFB from the camera aperture before pick and place operations.

1.4 SCOPE OF WORK

In this study, there are some works involved in making the program accomplish. First is to determine how the system of YOLO and Tensorflow software works. While doing that, basic Python language can be learned as it is used in order to execute the program. Then, the execution will show the image processing system by developing the image detection of the oil palm FFB. A hundred of images of the mentioned fruit will be used to test the running of the program. Next, the performance of state of the art Tensorflow system is tested on the Raspberry Pi device. The concept to find the distance from camera to oil palm FFB that has been detected is then studied in order for mechannization of pick and place to function.

CHAPTER II LITERATURE REVIEW

2.1 ECONOMIC PERFORMANCE OF OIL PALM FRUIT

Balu et al.[9] did an analysis on the industry performance of the oil palm planted area in Malaysia that has shown dramatic growth from a mere 55 000 ha in 1960 to 193 000 ha in 1970. The development was remarkable with the planted area reaching 1.02 million hectares in 1990 and further to 5.74 million hectares in 2016. In recent years, most of the expansion took place in Sabah and Sarawak due to declining availability of suitable land in Peninsular Malaysia. In 2016, about 47% of the planted area in Peninsular Malaysia, 27% in Sabah and 26% in Sarawak.

They also expected the adoption of mechanization to address the issue of labour shortage to increase oil palm productivity that ensure the sustainability of oil palm industry in Malaysia. The operation of oil palm plantation in the future are also expected to be more efficient, fully mechanized and automated. The use of drones, robotics, advanced sensors and digital technologies as well as user-friendly machineries and equipment are expected to minimize the manpower requirements.

This suits with the project of recognizing the image of the oil palm fruit itself by a camera to be built in the robot for pick and place operation purpose. This will cut the usage of labour, increase efficiency and also save time and cost in order to increase the capacity of oil palm fruit to be harvested or transported in the future. Figure 2.1 shows the increase in hectares through the years started from 1960 to 2016 across Malaysia.

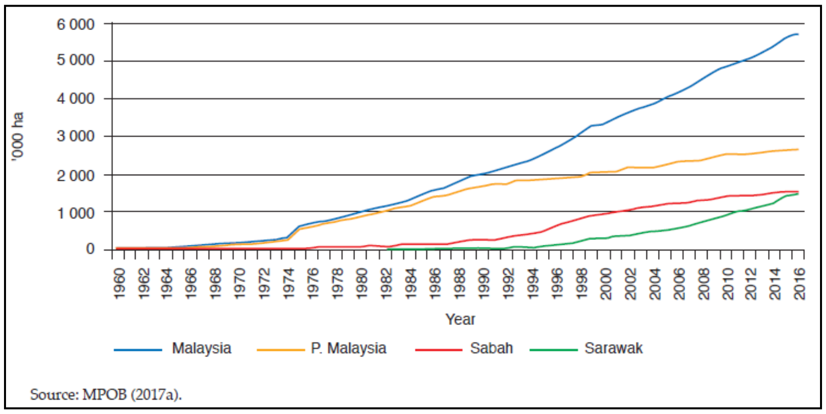


Figure 2.1: Oil palm planted area in Malaysia[9]

2.2 YOLO SOFTWARE

Redmon et al.[10] studied on YOLO (You Only Look Once) software, a new approach to object detection. The software frame object detection acts as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network estimates bounding boxes and class probabilities directly from full images in one evaluation. It can be optimized end-to-end directly on detection performance since the whole detection pipeline is a single network. The YOLO model processes images in real-time at 45 frames per second. YOLO model is also simple to construct and can be trained directly on full images.

In this research, YOLO software will be used in order to detect images of oil palm FFB. This is one application of development of AI in image processing. It is an easy approach as in earlier detection frameworks, looked at different parts of the image multiple times at different scales and repurposed image classification technique to detect objects. This approach is slow and inefficient. Therefore, YOLO takes a different approach by looking at the entire image only once and goes through the network once and detects objects which is very quick and chosen in this experiment. Figure 2.1 is the example of image detection using YOLO software of a dog, a bicycle and a truck in their own bounding box with labels on top of it.

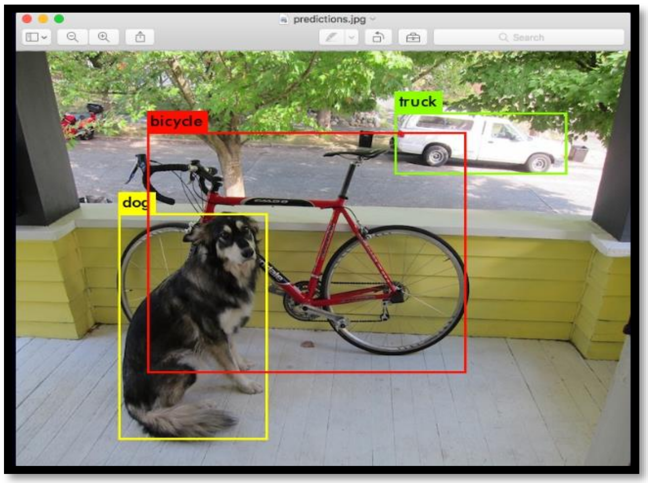


Figure 2.2: Image detection using YOLO [11]

YOLO is chosen because it is extremely fast. The neural network is run on a new image at test time to predict detections. Thus, the base network runs at 45 frames per second with no batch processing on a Titan X GPU and a fast version runs at more than 150 fps. This means streaming video can be processed in real-time with less than 25 milliseconds of latency. Furthermore, YOLO achieves more than twice the mean average precision of other real-time systems. Second, YOLO reasons globally about the image when making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance. Fast R-CNN, a top detection method [14], mistakes background patches in an image for objects because it cannot see the larger context. YOLO makes less than half the number of background errors compared to Fast R-CNN. Third, YOLO learns generalizable representations of objects. When trained on natural images and tested on artwork, YOLO outperforms top detection methods like DPM and R-CNN by a wide margin. Since YOLO is highly generalizable it is less likely to break down when applied to new domains or unexpected inputs. YOLO still lags behind state-of-the-art detection systems in accuracy. While it can quickly identify objects in images it struggles to precisely localize some objects, especially small ones.

2.3 TENSORFLOW

Tensorflow is a machine learning system [12],13] that operates by using dataflow graphs to represent computation, shared state, and the operations that mutate that state. The nodes of a dataflow graph are mapped across many machines in a cluster, and within a machine across many devices such as CPUs, GPUs, and custom designed ASICs known as Tensor Processing Units (TPUs). A graph is usually constructed using a front-end language such as Python[12]. TensorFlow also providess production prediction at scale, with the same models used for training. TensorFlow can train and run deep neural networks for image recognition which what this project will be focused on, recurrent neural networks, PDE (partial differential equation) based simulations and many more[14]. Figure 2.3 a cycle of how tensorflow algorithms works generally starting from a loss function until the output.

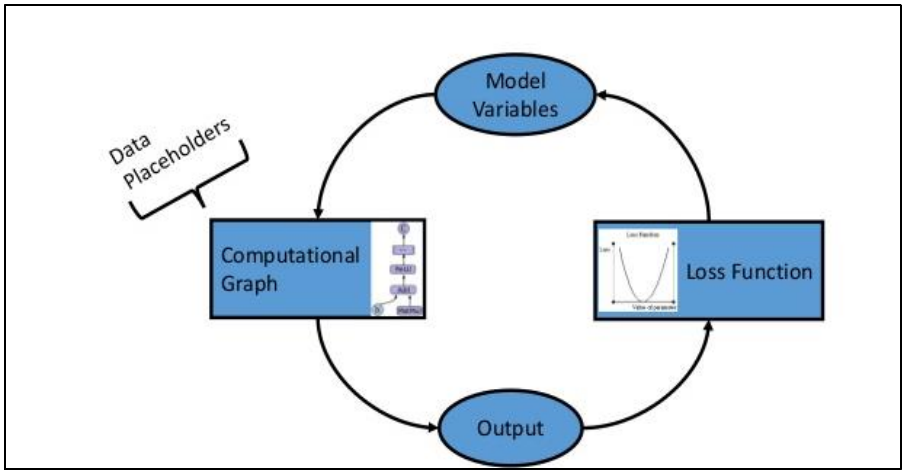


Figure 2.3: How Tensorflow Algorithms works[15]

2.4 IMAGE PROCESSING TECHNIQUES

Fadilah et al. [16] had undergone a study on image processing techniques of oil palm FFB by using a grading system. Their paper presents the application of color vision for automated ripeness classification of oil palm FFB. Images of oil palm FFBs of type DxP Yangambi were collected and analyzed using digital image processing techniques. Then the color features were extracted from those images and used as the inputs for Artificial Neural Network (ANN) learning. The performance of the ANN for ripeness classification of oil palm FFB was investigated using two methods: training ANN with full features and training ANN with reduced features based on the Principal Component Analysis (PCA) data reduction technique. The algorithm for the ripeness classification of oil palm FFB has been successfully implemented. The developed ripeness classifier can serve as a color sensor for automated oil palm FFB ripeness classification.

Shah and Kapdi [5] studied about the convolutional neural networks that achieved state-of-the-art performance on a number of image recognition benchmarks, including the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC-2012). The winning model on the localization sub-task was a network that predicts a single bounding box and a confidence score for each object category in the image. Such a model captures the whole-image context around the objects but cannot handle multiple instances of the same object in the image without naively replicating the number of outputs for each instance. They proposed a saliency-inspired neural network model for detection, which predicts a set of class-agnostic bounding boxes along with a single score for each box related to any object of interest. Based on the result obtained in their research, it can be concluded that the region based convolution neural network is more optimized at a very basic level. It is in dispute whether it can be said as the best form of solution to the problem or not. This result is valid only in a certain parameter.

The image processing technique of the study mentioned above can be implemented too in the experiment by using YOLO software. It is slightly different from the first image processing study because it is based on the grading system while the experiment is from the mesh of the graphics. The second study for the image processing technique is basically the application of the YOLO software that I am going to implement. The software can apply a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities[11]. The program that will be used looks at the whole image at test time so it can be informed by global context in the image by the predictions that has been made.

2.5 ULTRASONIC SENSOR FOR REVERSE PARKING APPLICATION

Road vehicles use parking sensors as proximity sensors to alert the driver of obstacles while parking[17]. Ultrasonic parking sensors use high-frequency sound waves to detect objects. These sensors produces sound pulses that reflect off of nearby objects. A receiver detects the reflected waves and calculates the distance from the vehicle to the object. Ultrasonic proximity sensors are connected to an alarm system that warns the driver of nearby obstacles with sounds. Advanced ultrasonic sensors translate to a pictograph on a vehicle’s infotainment screen that uses color blocks to represent the vehicle and possible obstacles.[18]

Moreover, backup cameras provides drivers to park more quickly and safely. Rear-facing cameras give the driver a much clearer and more accurate view of obstacles behind the car, and most backup systems include a warning tone when it is getting too close to an object.[19]

Besides, the addition of reversing camera systems which are combined with conventional parking aid systems ease the parking actions. The area behind the vehicle appears in the camera image on the monitor of the radio or navigation system. The display shows the driver whether there is anything in the way in real time. The distances measured by the ultrasonic sensors are embedded in the camera image in the form of coloured bars. This is to help the drivers to get the information at a glance.[20]

For the mechanization of the pick and place of the oil palm FFB, the same concept of reverse camera system can be used. The camera will detect the oil palm FFB and the image will appear in the camera as a real time view through ultrasonic sensors mounted at the truck. Based on Figure 2.5, it is how reverse camera system displayed on monitor of a Myvi car. For this project, oil palm FFB which has been detected will appear in the monitor and the distance will be shown using colour blocks to alert the operations of the pick and place of the bot.

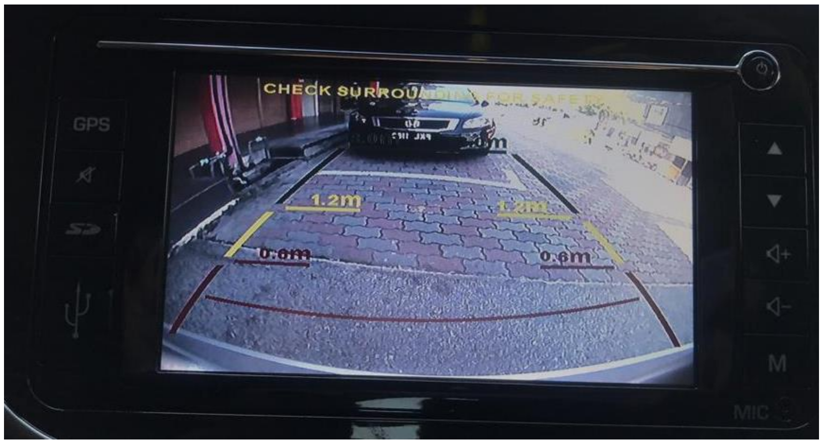


Figure 2.5: Reverse Camera system displayed on monitor of Myvi

1

CHAPTER III RESEARCH METHODOLOGY

3.1 APPLICATION

This chapter describes the programming environment that involves the application of object detection. The application used in this project is to identify the oil palm FFB at different conditions. Once oil palm FFB are detected, the application will use coloured bounding box to mark out the detected fruit on screen.

There are two versions of softwares/algorithms used in this project which are YOLO and Tensorflow. For YOLO, it displays extra feeds such as the coordinates of the bounding box, label coordinates and the confidence level. Confidence level is the probability of the trained object is in the image. In Tensorflow, the extra feed displayed is only the confidence level in the form of percentage. The trained datasets are then implemented in RaspberryPi to detect real-time oil palm fruit.

3.2 OBJECT RECOGNITION

Object recognition algorithms are required for identifying a specific oil palm fruit as an object in a digital image. In this project, object recognition algorithms are used based on supervised machine learning which uses training data with labels to learn a model of the data. Object recognition algorithms are implemented in OpenCV using Python as the programming language. Figure 3.2 shows the workflow of object detection algorithms starting from input image until its output of image recognition with frames, image label and confidence value.

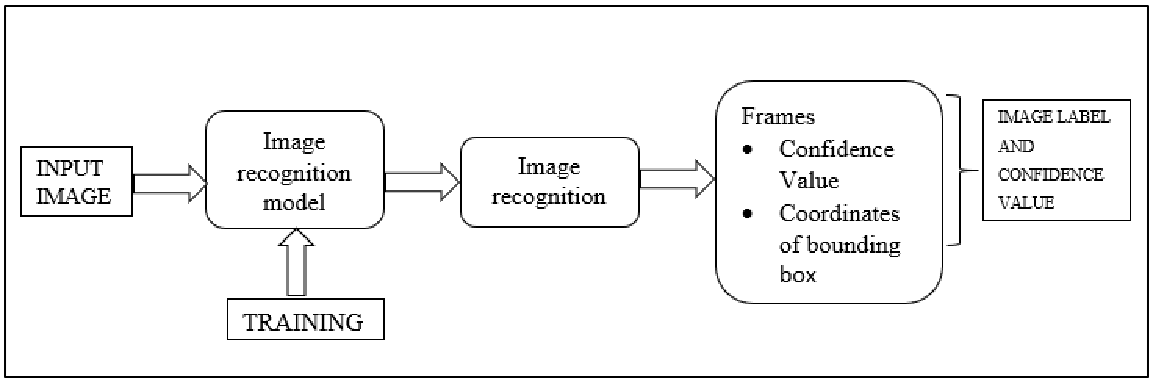


Figure 3.2: Workflow of algorithm

3.3 YOLO SET UP ON WINDOWS

Figure 3.3(a) shows the summary of the workflow of YOLO Object detetection to process images. The workflow starts with collection of samples images followed with its requirements as stated until the execution of Python script to process the image.

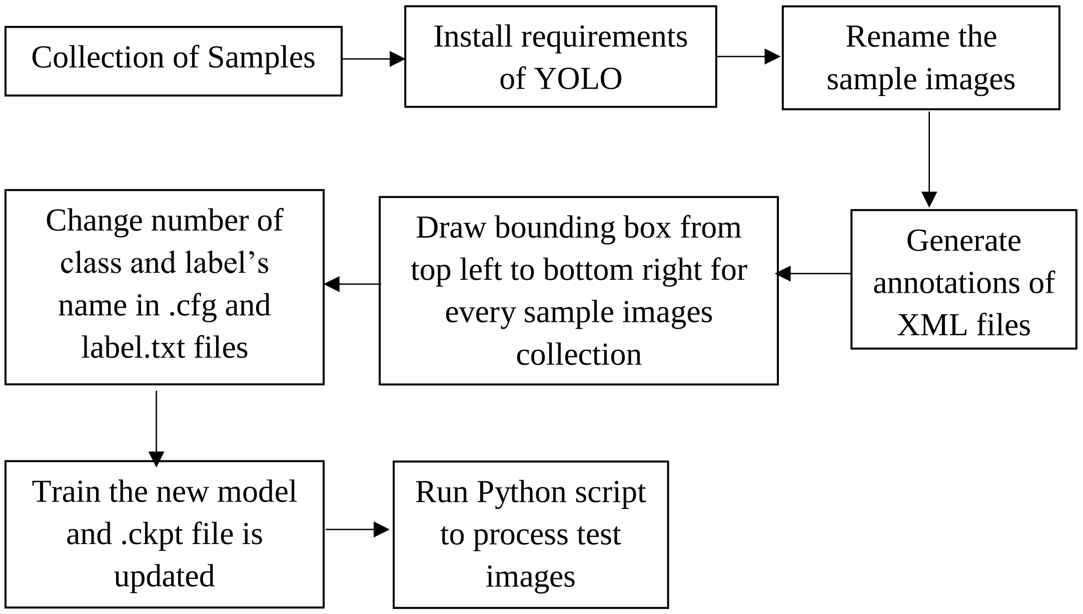

Collection of Samples
Install requirements of YOLO
Rename the sample images
Generate annotations of XML files
Draw bounding box from top left to bottom right for every sample images collection
Change number of class and label's name in .cfg and label.txt files
Train the new model and .ckpt file is updated
Run Python script to process test images

Figure 3.3(a): Workflow of YOLO Object Detection

Collection of Samples: An average of 700 sample images are gathered for each target object representing an oil palm fruit. Around 500 images are captured of a real oil palm fruit with different backgrounds and distances. Then, 200 images are of oil palm fruits are downloaded randomly from Google images. The images that are downloaded are ensured to be clear so the training will be effective. The number of sample images are varied in terms of distance, quantity and backgrounds because different images have different features. Figure 3.3(b) is the example of collection of sample images used for training. On the left side, it is the image of a single oil palm FFB. While on the right side shows the image of abundance of the respective fruits. Both types of images are used as training data sets.

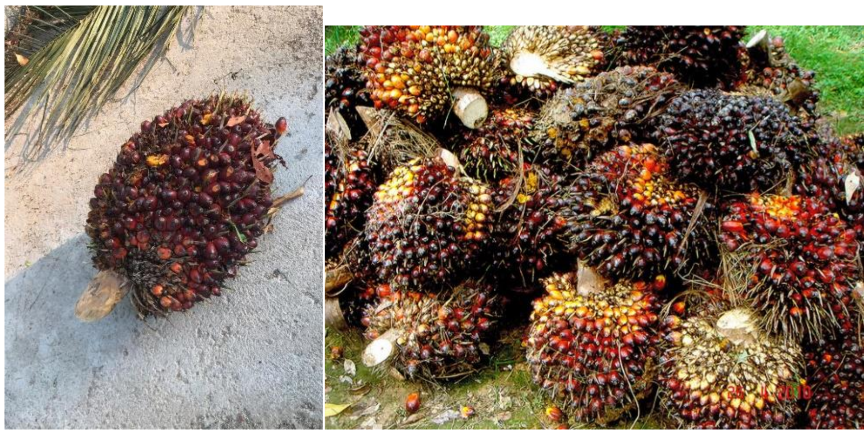


Figure 3.3(b): Example of oil palm pictures used for training

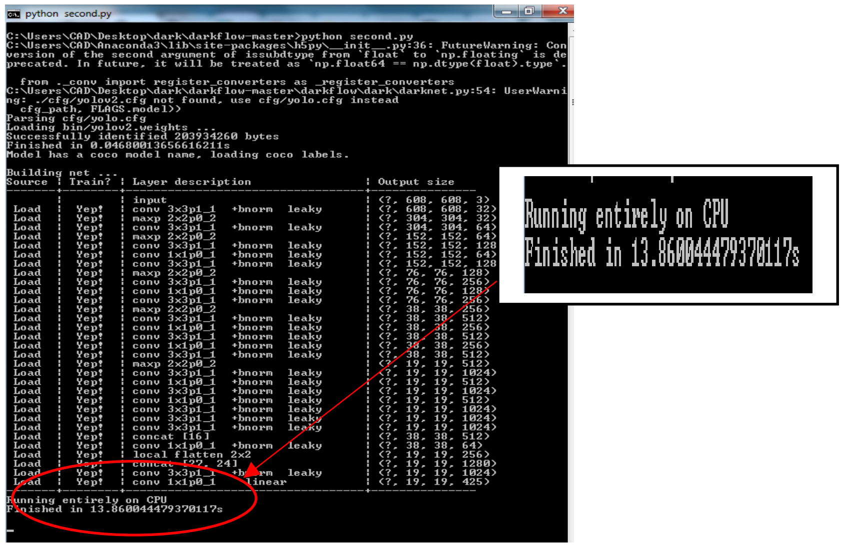
Requirements to run YOLO: In this experiment, Python 3.5 and 3.6 Anaconda are being used. Anaconda installation of Python has been done instead of traditional Python installation because it simplifies the install process. It includes ton of packages all at once. There is no need to waste time install and configure all the packages. Anaconda python is installed from website with the version Python 3.6. Once it is installed, the Anaconda Python are then setup.

Next, Tensorflow CPU version on Windows 10 is required to run YOLO. By running the script "*pip install tensorflow", it is installed.* OpenCV is also installed from a website of Unofficial Windows Binaries for Python Extension Packages. Darkflow Repository is then downloaded to run YOLO. YOLO is written in framework called Darknet which is a deep learning framework. A Tensorflow version of darknet is created called Darkflow. Darkflow is downloaded from github repository where a zip file is downloaded. Then, a directory is created in a folder located in Desktop named darkflow-master.

A library is build by opening a CMD window in the darkflow-master directory and python script are inserted: *pip install -e*. Lastly, weights for the model is downloaded. Weights of 608x608 are downloaded from YOLO official webpage. A new folder is created in the Darkflow-master directory called 'Bin'. The downloaded weights are put into the Bin folder.

3.3.1 Process image using trained weights from YOLO website

To test whether YOLO works or not, a random image is processed. A dog image is downloaded from Google images and saved it as dog.jpg in darkflow-master folder. Then, a Cmd window is opened in the folder and coding script to process image is run. The coding script to process image is shown in Appendix A. After running the command, the time taken to process the image are shown in the window command as in Figure 3.3.1(a). At the same time, a picture of dog with bounding box appeared showing that the image processing for YOLO works as shown in Figure 3.3.1(b).



```
C:\Users\CAD\Desktop\dark\darkflow-master>python second.py
C:\Users\CAD\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Con
version of the second argument of issubdtype from `float` to `np.floating` is de
precated. In future, it will be treated as `np.float64 == np.dtype(float).type`.

  from ._conv import register_converters as _register_converters
C:\Users\CAD\Desktop\dark\darkflow-master\darkflow\dark\darknet.py:54: UserWarni
ng: ./cfg/yolov2.cfg not found, use cfg/yolo.cfg instead
  cfg_path, FLAGS.model))
Parsing cfg/yolo.cfg
Loading bin/yolov2.weights ...
Successfully identified 203934260 bytes
Finished in 0.04680013656616211s
Model has a coco model name, loading coco labels.

Building net ...
Source | Train? | Layer description                | Output size
-------+--------+----------------------------------+---------------
       |        | input                            | (?, 608, 608, 3)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 608, 608, 32)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 304, 304, 32)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 304, 304, 64)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 152, 152, 64)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 152, 152, 128
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 152, 152, 64)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 152, 152, 128
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 76, 76, 128)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 76, 76, 256)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 76, 76, 128)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 76, 76, 256)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 38, 38, 256)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 38, 38, 512)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 38, 38, 256)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 38, 38, 512)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 38, 38, 256)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 38, 38, 512)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 19, 19, 512)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 19, 19, 512)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 19, 19, 512)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
 Load  |  Yep!  | concat [16]                      | (?, 38, 38, 512)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 38, 38, 64)
 Load  |  Yep!  | local flatten 2x2                | (?, 19, 19, 256)
 Load  |  Yep!  | concat [27, 24]                  | (?, 19, 19, 1280)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
 Load  |  Yep!  | conv 1x1p0_1    linear           | (?, 19, 19, 425)
-------+--------+----------------------------------+---------------
Running entirely on CPU
Finished in 13.860044479370117s

_
```

Figure 3.3.1(a): Result after running the command.

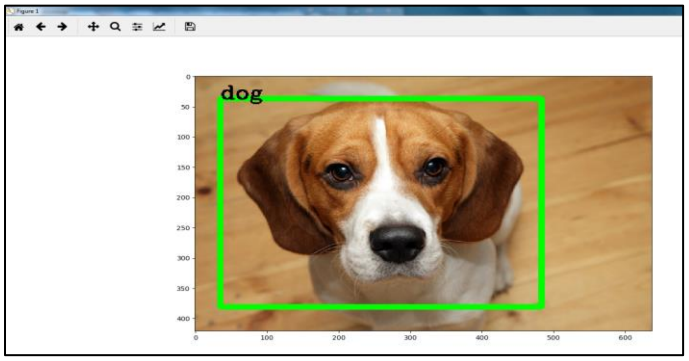
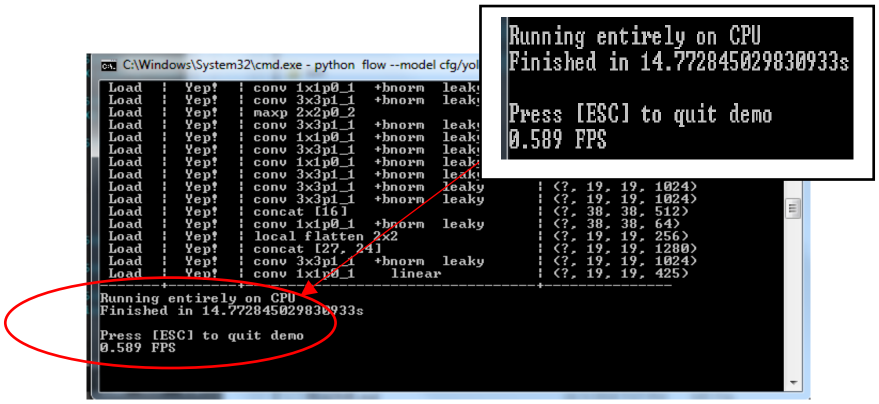


Figure 3.3.1(b): The dog image (Obtained from Google image) is then detected with bounding box and label.

3.3.2 Processing a video file using the trained weights from YOLO webpage

Same as process image stated before, a video processing can also be done using YOLO. A video of car video game is downloaded from Youtube in mp4 format and save in the Darkflow-master folder. Then, a Cmd window is opened in the folder and coding script to process video is run. The coding script to process video is shown in Appendix B. In the command window as shown in Figure 3.3.2(a), the finished time to process the video and the frame per second (FPS) are shown. Then, the video is processed showing bounding boxes on the trained images of the weights that has been downloaded before in Figure 3.3.2(b).



```
C:\Windows\System32\cmd.exe - python  flow --model cfg/yol
 Load  |  Yep!  | conv 1x1p0_1   +bnorm  leak
 Load  |  Yep!  | conv 3x3p1_1   +bnorm  leak
 Load  |  Yep!  | maxp 2x2p0_2
 Load  |  Yep!  | conv 3x3p1_1   +bnorm  leak
 Load  |  Yep!  | conv 1x1p0_1   +bnorm  leak
 Load  |  Yep!  | conv 3x3p1_1   +bnorm  leak
 Load  |  Yep!  | conv 1x1p0_1   +bnorm  leak
 Load  |  Yep!  | conv 3x3p1_1   +bnorm  leak
 Load  |  Yep!  | conv 3x3p1_1   +bnorm  leaky              | (?, 19, 19, 1024)
 Load  |  Yep!  | conv 3x3p1_1   +bnorm  leaky              | (?, 19, 19, 1024)
 Load  |  Yep!  | concat [16]                               | (?, 38, 38, 512)
 Load  |  Yep!  | conv 1x1p0_1   +bnorm  leaky              | (?, 38, 38, 64)
 Load  |  Yep!  | local flatten 2x2                         | (?, 19, 19, 256)
 Load  |  Yep!  | concat [27, 24]                           | (?, 19, 19, 1280)
 Load  |  Yep!  | conv 3x3p1_1   +bnorm  leaky              | (?, 19, 19, 1024)
 Load  |  Yep!  | conv 1x1p0_1     linear                   | (?, 19, 19, 425)
-------+--------+-------------------------------------------+----------------
Running entirely on CPU
Finished in 14.772845029830933s

Press [ESC] to quit demo
0.589 FPS
```

```
Running entirely on CPU
Finished in 14.772845029830933s

Press [ESC] to quit demo
0.589 FPS
```

Figure 3.3.2(a): Result of processing video using YOLO trained weights

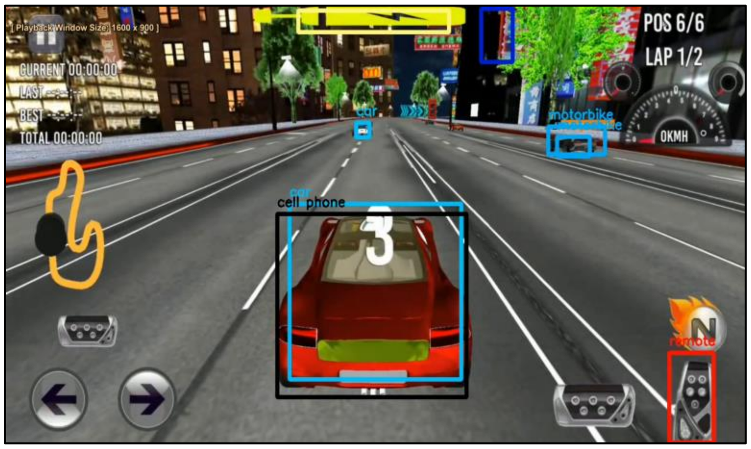


Figure 3.3.2(b): Bounding box and labels appeared in the video processed[10]

3.3.3 Train new models of oil palm fruit

To start training the new models which is the oil palm fruit, there are several steps to be followed. All the images collected are renamed with increasing numbers to synchronize for drawing the bounding box. They are renamed for easy tracking purpose when bounding box are drawn later. The Python script to rename the images are shown in Appendix C.

Next, annotation XML files are generated for each of the images collected as the training dataset. In this XML file, there are a bunch of information specifically the size of the bounding box drawn in the term of height and width all measured in pixels. To generate the file of each image, a script is being executed shown in Appendix D. Once the script is run, the collection of oil palm images popped up one by one, and bounding box are drawn at every image as shown in Figure 3.3.3(a). The bounding box are drawn from top left to bottom right as coordinates can be obtained from the script.

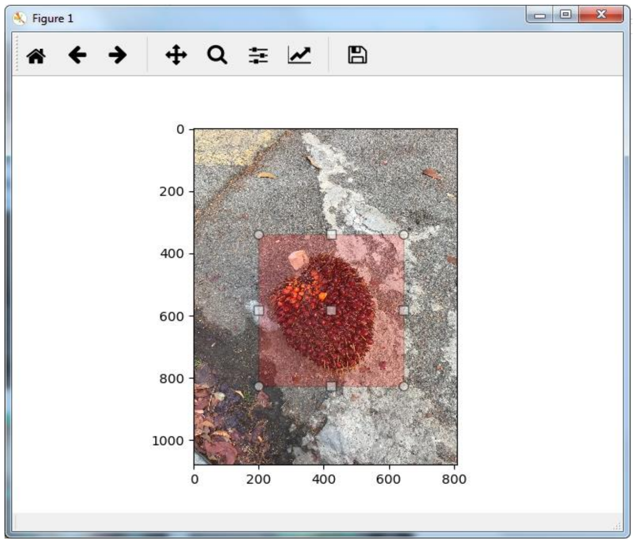


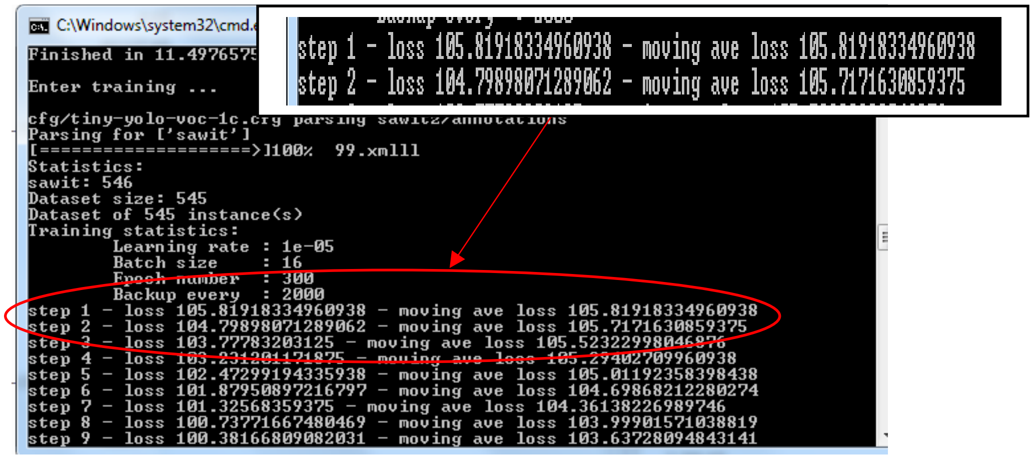
Figure 3.3.3(a): The red box shows the bounding box drawn

When all the images are drawn with bounding boxes where the oil palm fruits are located, model can now be trained. All the annotations XML files of every images are present with the coordinates of the bounding boxes. Next, weights can be loaded and model trained can be tested. A copy of configuration file tiny-yolo-voc.cfg is created and change the class of the region layer which is 1 because only one class type of oil palm fruit is available. In the tiny-yolo-voc-1c.cfg, filters in the convolutional layers is changed by num

* (classes + 5). Since the num is 5 and classes are 1 so 5 * (1 + 5) = 30. Therefore, the filters is changed to 30.

Labels.txt is changed where it includes the label(s) to train on (number of labels should be the same as the number of classes set in tiny-yolo-voc-1c.cfg file). In this case, labels.txt will contain only one label.

Lastly, a CMD window is opened in darkflow-master directory, flow --model cfg/tiny-yolo-voc-1c.cfg --load bin/tiny-yolo-voc.weights --train --annotation train/Annotations --dataset train/Images is typed and executed. This execution will train the dataset of the collected oil palm fruit images. Figure 3.3.3(b) shows dataset has started its training when the list of steps are shown in the command window.

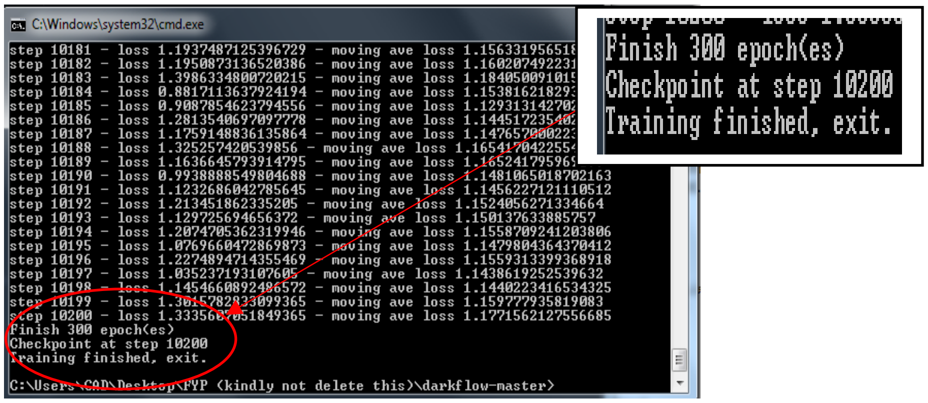


```
step 1 - loss 105.81918334960938 - moving ave loss 105.81918334960938
step 2 - loss 104.79898071289062 - moving ave loss 105.7171630859375
```

```
Finished in 11.4976575

Enter training ...

cfg/tiny-yolo-voc-1c.cfg parsing sawit2/annotations
Parsing for ['sawit']
[====================>]100%  99.xmlll
Statistics:
sawit: 546
Dataset size: 545
Dataset of 545 instance(s)
Training statistics:
        Learning rate : 1e-05
        Batch size    : 16
        Epoch number  : 300
        Backup every  : 2000
step 1 - loss 105.81918334960938 - moving ave loss 105.81918334960938
step 2 - loss 104.79898071289062 - moving ave loss 105.7171630859375
step 3 - loss 103.77783203125 - moving ave loss 105.52322998046876
step 4 - loss 103.231201171875 - moving ave loss 105.29402709960938
step 5 - loss 102.47299194335938 - moving ave loss 105.01192358398438
step 6 - loss 101.87950897216797 - moving ave loss 104.69868212280274
step 7 - loss 101.32568359375 - moving ave loss 104.36138226989746
step 8 - loss 100.73771667480469 - moving ave loss 103.99901571038819
step 9 - loss 100.38166809082031 - moving ave loss 103.63728094843141
```

Figure 3.3.3(b): The dataset is being trained.

During the training, the command window shows the statistics of the training, step by step with moving average loss. The traning will stop once it reached 300 epoch number. For every checkpoint, the data will be stored in a folder in the darkflow-master directory called ‘ckpt’. Figure 3.3.3(c) shows the end of training after 300 epoches that has been set and the checkpoint is at step 10200.



```
C:\Windows\system32\cmd.exe
step 10181 - loss 1.1937487125396729 - moving ave loss 1.156331956518
step 10182 - loss 1.1950873136520386 - moving ave loss 1.160207492231
step 10183 - loss 1.3986334800720215 - moving ave loss 1.184050091015
step 10184 - loss 0.8817113637924194 - moving ave loss 1.153816218293
step 10185 - loss 0.9087854623794556 - moving ave loss 1.129313142702
step 10186 - loss 1.2813540697097778 - moving ave loss 1.144517235402
step 10187 - loss 1.1759148836135864 - moving ave loss 1.147657000223
step 10188 - loss 1.325257420539856 - moving ave loss 1.165417042255
step 10189 - loss 1.1636645793914795 - moving ave loss 1.165241795969
step 10190 - loss 0.9938888549804688 - moving ave loss 1.1481065018702163
step 10191 - loss 1.1232686042785645 - moving ave loss 1.1456227121110512
step 10192 - loss 1.213451862335205 - moving ave loss 1.1524056271334664
step 10193 - loss 1.129725694656372 - moving ave loss 1.1501376338857 57
step 10194 - loss 1.2074705362319946 - moving ave loss 1.1558709241203806
step 10195 - loss 1.0769660472869873 - moving ave loss 1.1479804364370412
step 10196 - loss 1.2274894714355469 - moving ave loss 1.1559313399368918
step 10197 - loss 1.035237193107605 - moving ave loss 1.1438619252539632
step 10198 - loss 1.1454660892486572 - moving ave loss 1.1440223416534325
step 10199 - loss 1.3015782833099365 - moving ave loss 1.159777935819083
step 10200 - loss 1.3335607051849365 - moving ave loss 1.1771562127556685
Finish 300 epoch(es)
Checkpoint at step 10200
Training finished, exit.

C:\Users\CAD\Desktop\FYP (kindly not delete this)\darkflow-master>
```

Finish 300 epoch(es)
Checkpoint at step 10200
Training finished, exit.

Figure 3.3.3(c): Training stopped after 300 epoch(es)

3.4 TENSORFLOW SET UP ON WINDOWS

Same as YOLO, Tensorflow also requires Tensorflow and Anaconda Python environment to be set up. Next, TensorFlow Object Detection API repository is downloaded from GitHub of Edge Electronics. A folder is created directly in C: and name it “tensorflow1”. This working directory contains the full TensorFlow object detection framework, training images, training data, trained classifier, configuration files, and everything else needed for the object detection classifier. TensorFlow object detection repository is downloaded which is located at https://github.com/tensorflow/models and the zip file is downloaded. The downloaded zip file is opened and the “models-master” folder is extracted directly into the C:\tensorflow1 directory that is just created. The “models-master” is renamed to just “models”.

Then, Faster-RCNN-Inception-V2-COCO model is downloaded from TensorFlow's model zoo. TensorFlow provides several object detection models (pre-trained classifiers with specific neural network architectures) in its model zoo. Faster_rcnn_inception_v2_coco_2018_01_28 folder is extracted to the C:\tensorflow1\models\research\object_detection folder. From Figure 3.4, the object_detection folder should have all of these files.

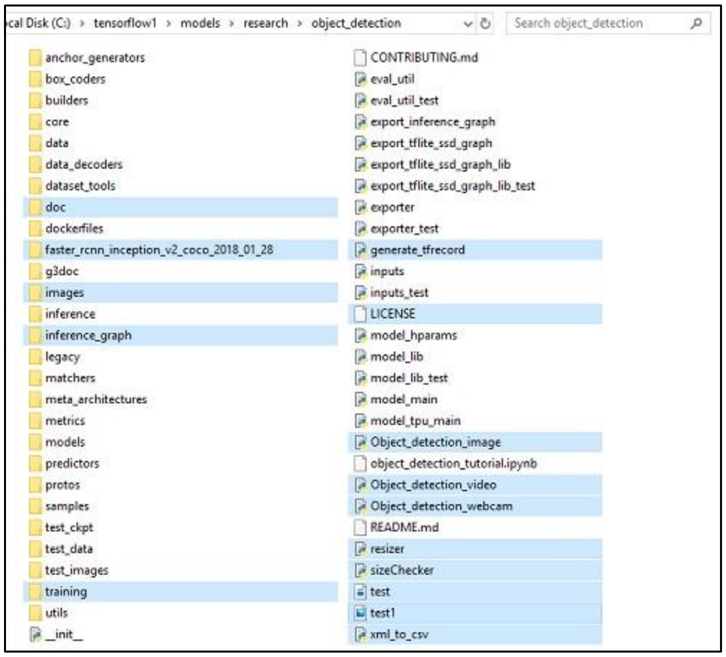


Figure 3.4: Files in object_detection folder

This repository contains the images, annotation data, .csv files, and TFRecords needed to train a playing card detector. It has scripts to test out the object detection classifier on images, videos, or a webcam feed.

3.4.1 Set up new Anaconda virtual environment

From the Start menu in Windows, Anaconda Prompt utility is searched and right click on it, and "Run as Administrator" is clicked. The command terminal will pop up and a new virtual environment called "tensorflow1" is created by issuing the following command:

```
C:\> conda create -n tensorflow1 pip python=3.6
```

Then, the environment is activated by issuing:

```
C:\> activate tensorflow1
```

Other necessary packages are installed by issuing the following commands:

```
(tensorflow1) C:\> conda install -c anaconda protobuf
(tensorflow1) C:\> pip install pillow
(tensorflow1) C:\> pip install lxml
(tensorflow1) C:\> pip install Cython
(tensorflow1) C:\> pip install jupyter
(tensorflow1) C:\> pip install matplotlib
```

```
(tensorflow1) C:\> pip install pandas
(tensorflow1) C:\> pip install opencv-python
```

3.4.2 Configure PYTHONPATH environment variable

A PYTHONPATH variable is created that points to the \models, \models\research, and \models\research\slim directories. The PYTHONPATH need to be set up everytime 'tensorflow1' virtual environment is excited. The following commands are issued from any directory:

```
(tensorflow1) C:\> set
PYTHONPATH=C:\tensorflow1\models;C:\tensorflow1\models\research;C:\tensorflow1
\models\research\slim
```

3.4.3 Compile Protobufs and Run setup.py

Protobuf files are compiled which are used by TensorFlow to configure model and training parameters. Every .proto file in the \object_detection\protos directory is called out individually by the command. In the Anaconda Command Prompt, directories are changed to the \models\research directory and the following command is run into the command line:

```
protoc --python_out=. .\object_detection\protos\anchor_generator.proto
.\object_detection\protos\argmax_matcher.proto
.\object_detection\protos\bipartite_matcher.proto
.\object_detection\protos\box_coder.proto
.\object_detection\protos\box_predictor.proto
.\object_detection\protos\eval.proto
.\object_detection\protos\faster_rcnn.proto
.\object_detection\protos\faster_rcnn_box_coder.proto
.\object_detection\protos\grid_anchor_generator.proto
.\object_detection\protos\hyperparams.proto
.\object_detection\protos\image_resizer.proto
.\object_detection\protos\input_reader.proto
.\object_detection\protos\losses.proto .\object_detection\protos\matcher.proto
.\object_detection\protos\mean_stddev_box_coder.proto
.\object_detection\protos\model.proto
.\object_detection\protos\optimizer.proto
.\object_detection\protos\pipeline.proto
.\object_detection\protos\post_processing.proto
.\object_detection\protos\preprocessor.proto
.\object_detection\protos\region_similarity_calculator.proto
.\object_detection\protos\square_box_coder.proto
.\object_detection\protos\ssd.proto
.\object_detection\protos\ssd_anchor_generator.proto
.\object_detection\protos\string_int_label_map.proto
.\object_detection\protos\train.proto
.\object_detection\protos\keypoint_box_coder.proto
```

```
.\object_detection\protos\multiscale_anchor_generator.proto
.\object_detection\protos\graph_rewriter.proto
```

Then, the following commands is run from the C:\tensorflow1\models\research directory:

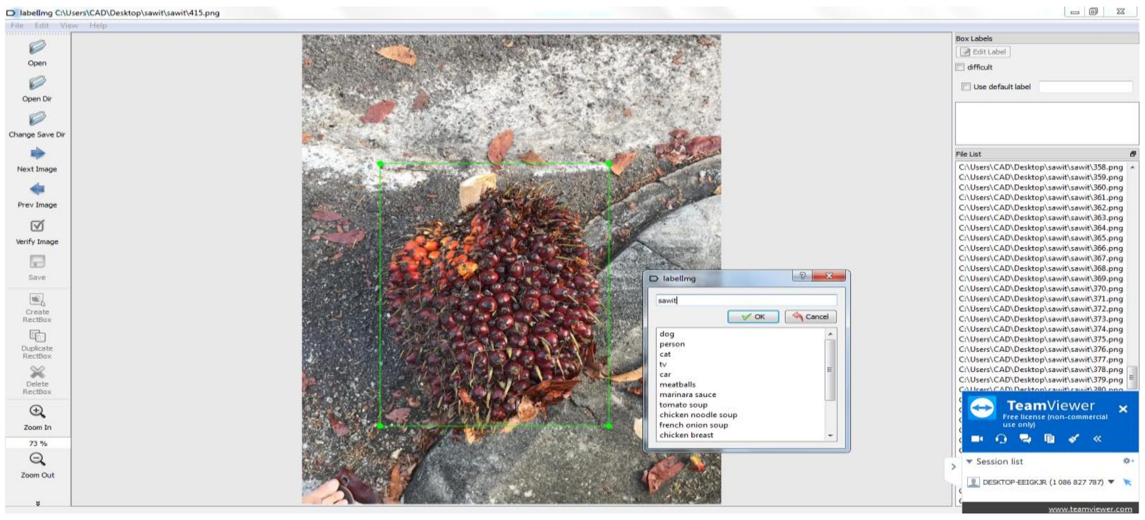
```
(tensorflow1) C:\tensorflow1\models\research> python setup.py build

(tensorflow1) C:\tensorflow1\models\research> python setup.py install
```

3.4.4 Gather Images

TensorFlow needs hundreds of images of an object to train a good detection. The training dataset are taken from the collection of samples from YOLO which total around 700 of oil palm fruit images. After all the images are ready, 20% of them are moved to the \object_detection\images\test directory, and 80% of them to the \object_detection\images\train directory. There are a variety of pictures in both the \test and \train directories.

3.4.5 Label Images

For drawing a bounding box around the oil palm fruit of every image, LabelImg is used as shown in Figure 3.4.5. It is downloaded online. It is pointed to \images\train directory, and then a box is drawn around each object in each image. The process is repeated for all the images in the \images\test directory.



sawit

```
dog
person
cat
tv
car
meatballs
marinara sauce
tomato soup
chicken noodle soup
french onion soup
chicken breast
```

```
C:\Users\CAD\Desktop\sawit\sawit\358.png
C:\Users\CAD\Desktop\sawit\sawit\359.png
C:\Users\CAD\Desktop\sawit\sawit\360.png
C:\Users\CAD\Desktop\sawit\sawit\361.png
C:\Users\CAD\Desktop\sawit\sawit\362.png
C:\Users\CAD\Desktop\sawit\sawit\363.png
C:\Users\CAD\Desktop\sawit\sawit\364.png
C:\Users\CAD\Desktop\sawit\sawit\365.png
C:\Users\CAD\Desktop\sawit\sawit\366.png
C:\Users\CAD\Desktop\sawit\sawit\367.png
C:\Users\CAD\Desktop\sawit\sawit\368.png
C:\Users\CAD\Desktop\sawit\sawit\369.png
C:\Users\CAD\Desktop\sawit\sawit\370.png
C:\Users\CAD\Desktop\sawit\sawit\371.png
C:\Users\CAD\Desktop\sawit\sawit\372.png
C:\Users\CAD\Desktop\sawit\sawit\373.png
C:\Users\CAD\Desktop\sawit\sawit\374.png
C:\Users\CAD\Desktop\sawit\sawit\375.png
C:\Users\CAD\Desktop\sawit\sawit\376.png
C:\Users\CAD\Desktop\sawit\sawit\377.png
C:\Users\CAD\Desktop\sawit\sawit\378.png
C:\Users\CAD\Desktop\sawit\sawit\379.png
C:\Users\CAD\Desktop\sawit\sawit\380.png
```

DESKTOP-EEIGKJR (1 086 827 787)

Figure 3.4.5: LabelImg application to label and draw bounding box of object

LabelImg saves a .xml file containing the label data for each image. These .xml files are used to generate TFRecords, which are one of the inputs to the TensorFlow trainer. Thus, there is one .xml file for each image in the \test and \train directories.

3.4.6 Generate Training Data

With the images labeled, TFRecords are generated to serve as input data to the TensorFlow training model. First, the image .xml data are used to create .csv files containing all the data for the train and test images. From the \object_detection folder, following command is entered in the Anaconda command prompt:

```
        (tensorflow1)   C:\tensorflow1\models\research\object_detection>   python
xml_to_csv.py
```

This creates a train_labels.csv and test_labels.csv file in the \object_detection\images folder. Then, TFRecord files are generated by issuing these commands from the \object_detection folder:

```
python generate_tfrecord.py --csv_input=images\train_labels.csv --
image_dir=images\train --output_path=train.record

python generate_tfrecord.py --csv_input=images\test_labels.csv --
image_dir=images\test --output_path=test.record
```

These generate a train.record and a test.record file in \object_detection. These will be used to train the new object detection classifier.

3.4.7 Label map

The label map tells the trainer what each object is by defining a mapping of class names to class ID numbers. A text editor is used to create a new file and save it as labelmap.pbtxt in the C:\tensorflow1\models\research\object_detection\training folderIn the text editor, copy or type in the label map in the format below:

```
item {
  id: 1

  name: 'sawit'
```

3.4.8 Configure Training

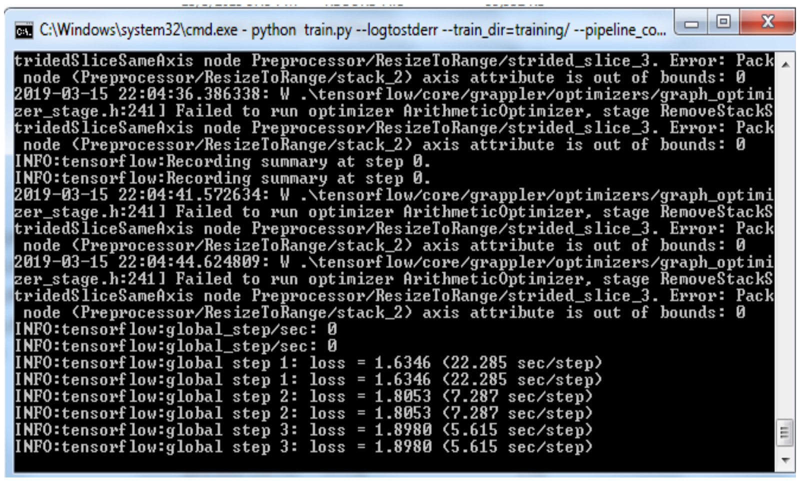
Finally, the object detection training pipeline is configured. It defines which model and what parameters will be used for training. This is the last step before running training. C:\tensorflow1\models\research\object_detection\samples\configs is navigated and the faster_rcnn_inception_v2_pets.config file is copied into the \object_detection\training directory. Then, the file is opened with a text editor.

3.4.9 Run the Training

From the \object_detection directory, the following command is issued to begin training:

```
python train.py --logtostderr --train_dir=training/ --
pipeline_config_path=training/faster_rcnn_inception_v2_pets.config
```

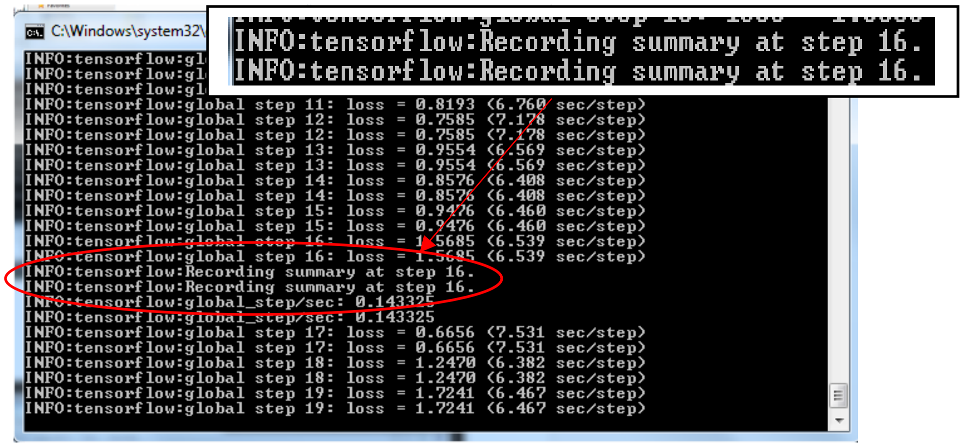
If everything has been set up correctly, TensorFlow initialize the training. The initialization take up to 30 seconds before the actual training begins. When training begins, the command windows will show ouput as shown in Figure 3.4.9(a). During the training, the command windows shows the recoding summary as in Figure 3.4.9(b).



C:\Windows\system32\cmd.exe - python train.py --logtostderr --train_dir=training/ --pipeline_co...

```
tridedSliceSameAxis node Preprocessor/ResizeToRange/strided_slice_3. Error: Pack
 node (Preprocessor/ResizeToRange/stack_2) axis attribute is out of bounds: 0
2019-03-15 22:04:36.386338: W .\tensorflow/core/grappler/optimizers/graph_optimi
zer_stage.h:241] Failed to run optimizer ArithmeticOptimizer, stage RemoveStackS
tridedSliceSameAxis node Preprocessor/ResizeToRange/strided_slice_3. Error: Pack
 node (Preprocessor/ResizeToRange/stack_2) axis attribute is out of bounds: 0
INFO:tensorflow:Recording summary at step 0.
INFO:tensorflow:Recording summary at step 0.
2019-03-15 22:04:41.572634: W .\tensorflow/core/grappler/optimizers/graph_optimi
zer_stage.h:241] Failed to run optimizer ArithmeticOptimizer, stage RemoveStackS
tridedSliceSameAxis node Preprocessor/ResizeToRange/strided_slice_3. Error: Pack
 node (Preprocessor/ResizeToRange/stack_2) axis attribute is out of bounds: 0
2019-03-15 22:04:44.624809: W .\tensorflow/core/grappler/optimizers/graph_optimi
zer_stage.h:241] Failed to run optimizer ArithmeticOptimizer, stage RemoveStackS
tridedSliceSameAxis node Preprocessor/ResizeToRange/strided_slice_3. Error: Pack
 node (Preprocessor/ResizeToRange/stack_2) axis attribute is out of bounds: 0
INFO:tensorflow:global_step/sec: 0
INFO:tensorflow:global_step/sec: 0
INFO:tensorflow:global step 1: loss = 1.6346 (22.285 sec/step)
INFO:tensorflow:global step 1: loss = 1.6346 (22.285 sec/step)
INFO:tensorflow:global step 2: loss = 1.8053 (7.287 sec/step)
INFO:tensorflow:global step 2: loss = 1.8053 (7.287 sec/step)
INFO:tensorflow:global step 3: loss = 1.8980 (5.615 sec/step)
INFO:tensorflow:global step 3: loss = 1.8980 (5.615 sec/step)
```

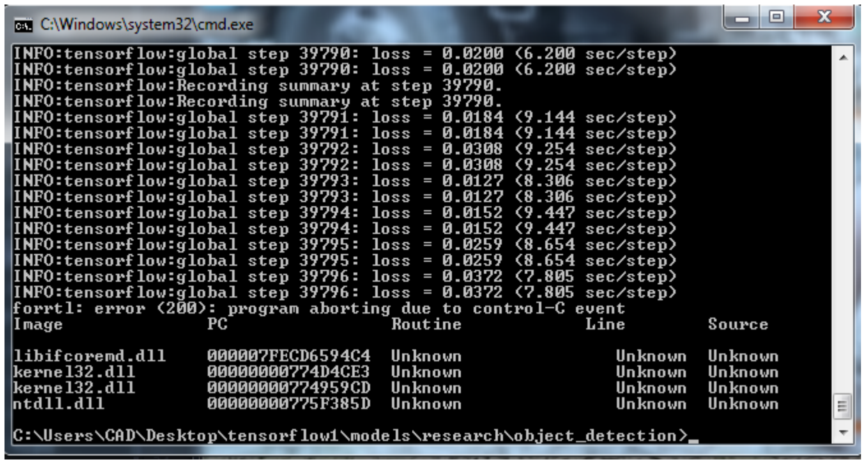
Figure 3.4.9(a): The command window shows the training started



```
C:\Windows\system32\
INFO:tensorflow:gl
INFO:tensorflow:gl
INFO:tensorflow:gl
INFO:tensorflow:global step 11: loss = 0.8193 (6.760 sec/step)
INFO:tensorflow:global step 12: loss = 0.7585 (7.178 sec/step)
INFO:tensorflow:global step 12: loss = 0.7585 (7.178 sec/step)
INFO:tensorflow:global step 13: loss = 0.9554 (6.569 sec/step)
INFO:tensorflow:global step 13: loss = 0.9554 (6.569 sec/step)
INFO:tensorflow:global step 14: loss = 0.8576 (6.408 sec/step)
INFO:tensorflow:global step 14: loss = 0.8576 (6.408 sec/step)
INFO:tensorflow:global step 15: loss = 0.9476 (6.460 sec/step)
INFO:tensorflow:global step 15: loss = 0.9476 (6.460 sec/step)
INFO:tensorflow:global step 16: loss = 1.5685 (6.539 sec/step)
INFO:tensorflow:global step 16: loss = 1.5685 (6.539 sec/step)
INFO:tensorflow:Recording summary at step 16.
INFO:tensorflow:Recording summary at step 16.
INFO:tensorflow:global_step/sec: 0.143325
INFO:tensorflow:global_step/sec: 0.143325
INFO:tensorflow:global step 17: loss = 0.6656 (7.531 sec/step)
INFO:tensorflow:global step 17: loss = 0.6656 (7.531 sec/step)
INFO:tensorflow:global step 18: loss = 1.2470 (6.382 sec/step)
INFO:tensorflow:global step 18: loss = 1.2470 (6.382 sec/step)
INFO:tensorflow:global step 19: loss = 1.7241 (6.467 sec/step)
INFO:tensorflow:global step 19: loss = 1.7241 (6.467 sec/step)
```

```
INFO:tensorflow:Recording summary at step 16.
INFO:tensorflow:Recording summary at step 16.
```

Figure 3.4.9(b): The command window shows the recording summary



```
C:\Windows\system32\cmd.exe
INFO:tensorflow:global step 39790: loss = 0.0200 (6.200 sec/step)
INFO:tensorflow:global step 39790: loss = 0.0200 (6.200 sec/step)
INFO:tensorflow:Recording summary at step 39790.
INFO:tensorflow:Recording summary at step 39790.
INFO:tensorflow:global step 39791: loss = 0.0184 (9.144 sec/step)
INFO:tensorflow:global step 39791: loss = 0.0184 (9.144 sec/step)
INFO:tensorflow:global step 39792: loss = 0.0308 (9.254 sec/step)
INFO:tensorflow:global step 39792: loss = 0.0308 (9.254 sec/step)
INFO:tensorflow:global step 39793: loss = 0.0127 (8.306 sec/step)
INFO:tensorflow:global step 39793: loss = 0.0127 (8.306 sec/step)
INFO:tensorflow:global step 39794: loss = 0.0152 (9.447 sec/step)
INFO:tensorflow:global step 39794: loss = 0.0152 (9.447 sec/step)
INFO:tensorflow:global step 39795: loss = 0.0259 (8.654 sec/step)
INFO:tensorflow:global step 39795: loss = 0.0259 (8.654 sec/step)
INFO:tensorflow:global step 39796: loss = 0.0372 (7.805 sec/step)
INFO:tensorflow:global step 39796: loss = 0.0372 (7.805 sec/step)
forrtl: error (200): program aborting due to control-C event
Image              PC                Routine            Line        Source

libifcoremd.dll    000007FECD6594C4  Unknown               Unknown  Unknown
kernel32.dll       00000000774D4CE3  Unknown               Unknown  Unknown
kernel32.dll       00000000774959CD  Unknown               Unknown  Unknown
ntdll.dll          00000000775F385D  Unknown               Unknown  Unknown

C:\Users\CAD\Desktop\tensorflow1\models\research\object_detection>_
```

Figure 3.4.9(c): The command window shows the completion of training

Each step of training reports the loss. It will start high and get lower and lower as training progresses. For the training on the Faster-RCNN-Inception-V2 model, it started from loss at about 1.6 and let it dropped below 0.03, which will take about 39796 steps. Figure 3.4.9(c) shows the completion of the training with the number of steps stated.

3.4.10 Export Inference Graph

The last step is to generate the frozen inference graph (.pb file). From the \object_detection folder, the following command is entered where “XXXX” in “model.ckpt-XXXX” is replaced with the highest-numbered .ckpt file in the training folder:

```
python export_inference_graph.py --input_type image_tensor --
pipeline_config_path training/faster_rcnn_inception_v2_pets.config --
```