TALLINN UNIVERSITY OF TECHNOLOGY
SCHOOL OF ENGINEERING
Department of Electrical Power Engineering and Mechatronics

Vision System for Apple Harvesting Robot

Masinnägemise süsteem õuna koristamise robotile

MASTER THESIS

Student:	JESIL KOCHIKKARAN BADHARUDHEEN
Student code:	194309MAHM
Supervisor:	Märt Juurma Junior researcher, Department Of Electrical Power Engineering and Mechatronics

Tallinn 2021

(*On the reverse side of title page*)

AUTHOR’S DECLARATION

Hereby I declare, that I have written this thesis independently.
No academic degree has been applied for based on this material. All works, major viewpoints and data of the other authors used in this thesis have been referenced.

“.......” .... 20…..

Author: ....
/signature /

Thesis is in accordance with terms and requirements

“.......” .... 20….

Supervisor: …........................
/signature/

Accepted for defence

“.......”....................20… .

Chairman of theses defence commission: ....
/name and signature/

Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Jesil Kochikkaran Badharudheen

1. grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis Vision system for apple harvesting robot,

supervised by, Märt Juurma

1.1 to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;

1.2 to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.

2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.

3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

______________ (date)

Department of Electrical Power Engineering and Mechatronics

[1] *The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.*

THESIS TASK

Student: Jesil Kochikkaran Badarudheen 194309 MAHM

Study programme, MAHM02/18 - Mechatronics

main speciality: Mechatronics

Supervisor(s): Mart Juurma, Junior Researcher

Thesis topic:

(in English) Vision System for Apple Harvesting Robot

(in Estonian) Masinnägemise süsteem õuna koristamise robotile

Thesis main objectives:

1. Perform background research on existing methods and determine the suitable approach.
2. Gathering a deep understanding of the equipment that needs to be used in this project. Choose the most advanced method for object detection and localization.
3. Improve the accuracy of the method of apple detection, and make theharvesting time up to the desired interval.

Thesis tasks and schedule:

No	Task	Month
1	Literature review	December
2	Development methodology	January
3	Research on the equipment's and software platforms required for the work	February - March
4	Design and development of fruit detector using machine vision	March-April
5	Thesis defense	May

Language: English **Deadline for submission of thesis:** "18th" May 2021 a

Student: Jesil Kochikkaran Badharudheen .... "......."............20....a

/signature/

Supervisor: Mart Juurma .... "......." ....20....a

/signature/

Consultant: .... "......." ....20....a

/signature/

Head of study programme: Prof. Dr. Mart Tamre .... "......."......20.....a

/signature/

Terms of thesis closed defence and/or restricted access conditions to be formulated on the reverse side.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

Harvesting robots for real-time applications are the propitious future of the agricultural industry. Even so, there are still many hurdles to overcome in developing a fully functioning robotic harvesting system. One of the most crucial aspects of these problems is vision. In real-world applications, traditional vision approaches often have flaws in terms of precision, accuracy, and performance. Detecting fruits in natural environments is difficult due to occlusion between leaves and fruits, clustering, and shadowing. The proposed method using the YOLO state-of-the-art object detection method is suitable to overcome this challenge. The task includes the detection of Matured and immature apples in color images acquired by a camera using the YOLO detection algorithm. This thesis is provided by the Department of Electrical and Power Engineering and Mechatronics of Tallinn University of Technology. I'm grateful for the opportunity to work on such a fascinating field as object detection with deep learning.

Here, the author would like to express his deepest gratitude to the supervisor Märt Juurma, Junior researcher Institute Of Electrical Power Engineering and Mechatronics who proposed this thesis and helped me find answers to all of my questions and deal with the problems that arose. I accomplished the mission under his guidance and learned a great deal of new information.

Finally, I would like to thank, Prof. Mart Tamre for the proper guidelines in writing the thesis and help to access the lab at a crucial time. Special thanks to the mechatronics department for the assistance.

Keywords: Machine vision, Deep learning, Object detection, YOLO, Darknet, Google colab, LabeIimg.

List of abbreviations and symbols

RGB – Red, Green, Blue color space for digital images

CNN- Convolutional neural networks

RNN – Recurrent neural network

SSD – single shot detector

YOLO – You only look once

HOG- Histogram of oriented gradients

LBP – Local binary pattern

R-CNN – Region based convolutional neural network

ROI – Region of interest

SVM – Support vector machine

Map – Mean average precision

WRC – Weighted-residual connections

CSP – Cross-stage partial connections

CmBN – Cross mini-batch Normalization

CUDA – Compute unified device architecture

GPU – Graphics processing unit

IoU – Intersection over union

NMS – Non-maximum suppression

CuDNN- GPU accelerated library for deep learning

GitHub- Version-control and collaboration platform

1. INTRODUCTION

1.1 Overview

Agriculture is both a major industry and the backbone of the economy. Factors such as climate change, population growth, and food security issues have propelled the sector to pursue more creative ways of preserving and improving crop yields. Consequently, AI is gradually evolving as part of the technical evolution of the industry [1]. The vast verities of data collecting devices help the agriculture for the high-tech revolution especially in the part of precision farming. The internet of things era leads a way to connect everything in our lives can be linked to a wi-fi connection and the same applies to agriculture and farming. Machine learning is possible for any of the machines, even fertilizer application can be trained to apply only for the required plants, for example, A robot that can constantly pluck berries has appeal, saving farmers productand money.

Future farming has become more sophisticated. Digitized crop cultivation, robots, and drones will play an important part in farming in the future**.** The agriculture industry has many problems, including the decreasing number of farmworkers and the increasing costof fruit harvesting. AI is getting traction in the agricultural industry and is steadily being merged into robotics developed in this sector. To perform important agricultural tasks such as harvesting crops at a higher leveland quicker speed than human workers, industries are designing and programming autonomous robots. In an initiative to further address challenges in the labor market, technology is also emerging. From 2014 to 2024, the industry is expected to witness a 6 percent reduction in agricultural workers [2].

Robots are also of interest to growers of fruit and vegetables that are currently picked byhand, which would be a lot faster and cheaper if they were automated. Currently, there aren’t enough workers to continuously pick every apple (or any fruits or vegetables) thatevery plant produces. Especially in this kind of pandemic situation. The present ranchers face loads of difficulties: a maturing labor force, a lack of ease of work, ecological perils, and environmental change, to give some examples [3]. Furthermore, for each issue, there is by all accounts a robot or automated gadget in progress to fix it. It gives them [farmers] the authorization to be inventive, the capacity to be innovative with their gear. It enables the ranchers to reclaim responsibility for innovation.

1.2 Problem statement

Harvesting activity is a tedious and work-concentrated cycle in the creation of foods grown from the ground, which is likewise a significant piece of the entire agricultural creation measure. The absence of agrarian work may cause the disappointment of convenient harvesting, hence making colossal misfortunes in rural creation. The design of apple trees has developed throughout the long term, and it's currently regular to develop them on lattices like y tomatoes or cucumbers. Current apple trees are additionally more modest, got from dwarf varietals that yield more per section of land and produce an organic product all the more rapidly after being planted. These green jumps have permitted ranchers to twofold their apple yields. They have likewise made the work of picking simpler for people and now, for robots. Apple harvesting could be a suitable task for a robot system. This however requires a new approach to how the trees are grown, placed and which robotics designs would work with that. Currently, there are not many robotic systems that can safely harvest apples. Some systems bump the tree and pick the applesfrom the ground or employ a canvas cone that wraps around the tree stem for gathering the apples. This however is not very desirable, because the apples will receive falling damage and their value is reduced.

1.3 Objectives of the research

The objective of this thesis is to develop an apple detection and localization system for an apple harvesting robot. As further presented in the thesis, the main objectives of this work are:

I. Research on the current methods and technologies and find out the latest techniques to detect the apples, analyze the designfor a harvesting mechanism.
II. A highly fast and accurate method must be used herein to detect the position of fruit.
III. Develop an algorithm to Predict the apple whether it’s ripe or not.
IV. Test the algorithm over images, videos, and real-time scenarios.
V. Verify that the objective is met.

The improvement of the control strategy increases the operating efficiency ofthe robot. Developing an apple-picking robot’s vision system has been an interesting and exciting challenge combining multiple domains of expertise such as machine learning, robotics, and mechanical engineering. Through this prospect, the author can implement his knowledgethat he acquired during this course**.** Apple harvesting could be a suitable task for a robot system. This however requires a new approach to how the trees are grown, placed and which robotics designs would work with that. An assortment of models has been created. Nonetheless, the current apple reaping robot model is as yet in the trial research stage because of its low productivity. This mechanism can be applied even if it isa near species of apple, moreover one learns again with the target fruit, harvesting fruits,such as pears is highly possible**.** The fast, precise recognition and localization of the targetfruit can overcome the bottleneck of low efficiency of visual system operation.

Chapter 1 provides an introduction to the thesis followed by the motivation and objectives of the study. The objectives are further detailed in the body of the report.

Chapter 2 contains an analysis of previous studies conducted related to the objectives of the thesis referring to the published data. the section further discusses the justification of the scope of the thesis.

Chapter 3 consists of the methodology of the thesis where it describes the proposed solution, selection of the hardware and software that will be used in the project. Furthermore, it provides details on three conceptual approaches to the solution and selection of the most suitable approach.

Chapter 4 comprises a detailed explanation of how the selected approach is developed. This includes the process flow chart, the model development methods used, training and testing

Chapter 5 provides the details on the limitation of the projects, justifications of the project, suggestions for further developments, and a summary of the project.

2. LITERATURE REVIEW AND BACKGROUND

This chapter describes the existing solutions and research done on the topic. The existing literature will be further reviewed in this section, describing the previous approaches thatare closely related to the intent of this study and will use the principles to identify a possiblesolution for the harvesting system.

The problem will be described and discussed in this section. The agriculture industry has many problems, the labor shortage has been accompanied by higher wages. Ranchers have additionally been battered by trade wars, which have made fares droop, and they face dangers from weather patterns attached to environmental change. Robots could help convey a portion of the heaviness of these issues. The improvement of the control strategy increases theoperating efficiency of the robot. The advent of autonomous system architectures allows us to develop a completely new range of agricultural equipment based on small smart machines that can do the right thing, in the right place, at the right time in the right way [4].

Apple harvesting could be a suitable task for a robot system. This however requires a new approach to how the trees are grown, placed and which robotics designs would work withthat. The development of an automated fruit harvesting robot is a viable solution to theseproblems. The automated harvesting consists of two stages;

I. Image acquisition using sensor and detection and localization of fruit using computer vision.

II. Detect the position of the fruit and harvest the fruit using manipulators and end-effector.

2.1 Computer vision and detection

The first step is to acquire the images of the fruit from the tree. Then the information regarding whether the apple is ripe or not the number of apples and its position is acquired from a vision system. Then the data about the position is sent to the manipulator for harvesting the fruit from a tree. A machine vision system for fruit detection and localizationinvolves various steps of data collection and processing as discussed below.

One of the essential undertakings inside a machine vision framework is image acquisition. Based on machine vision various researchers have studied for the detection and localization of fruits. In the Gongal et al. review [5]. Inthese approaches, color, spectral, or thermal cameras have been commonly used. In addition to geometric and texture information, color cameras provide opportunities for basic color-based fruit segmentation with images acquired. The spectral camera providesspectral information, along with spatial object information. Spectral imaging, even if the fruit and background color are identical, has the potential to detect fruit. Thermal camerascapture the object's temperature signature, based on the temperature contrast betweenthe fruit and the background, thermal imaging has been used to detect fruit. The precisionof this method was limited by the fruit size and direct exposure to sunlight.

For the classification of fruits, Bulanon et al. used luminance and RGB color difference in differentlighting conditions [6]. Apple detection was carried out and an accuracy of 88.0 percent was achieved under controlled lighting conditions based on the red color difference between the items. The author accepted that the precision of the detection of fruit depends on the lighting conditions during photography and that high accuracy of the detection of apples can be obtained under controlled lighting conditions. Zhou et al. (2012) used colorfeatures in both RGB and HSI space for apple detection [7]. The distinction between red, green, and blue shading channels was utilized as an action to section red and green apples from the background. The methods [6] [7] takes the color difference between the fruits and leaves is very small and easy to be changed by illuminations. Thanks to the special hue, the identification of red fruits is comparatively simpler by machine vision, so needs to concentrate on other aspects except color. As a measure for segmenting red and greenapples from the backdrop, the differentiation between RGB shading channels was utilized. Besides, to section red apples, a threshold value in the saturation channel was alsoused.

The quality of images that later impact the performance of algorithms for fruit detection is impacted by the color saturation inconsistency. Similarly, monochrome cameras lack color information, while spectral cameras either require considerable time to run in real-time or have a narrow operational range during the day [5]. Each of those sensors, on the other hand, has its strengths. Monochrome cameras are less sensitive than color cameras to lighting changes, whereas color cameras have color detail, which is one of themost used fruit detection features. In various spectrum ranges, the spectral camera can have information and can also detect fruit independently of its color [5].

Linker et al. [8] Many features have been introduced to improve the accuracy of fruit detection methods. Different image recognition methods may also be carried out for fruitidentification using a color camera. Researchers have planned vision frameworks to secure pictures at dusk [8] and evening [9] to limit the impacts of direct daylight and variable lighting conditions.

A supporting over-the-row device that could address possible problems with image acquisition under field conditions was developed and manufactured [10]. This supporting stage has artificial lights that work with uniform picture openness with semi-controlled lighting. The accuracy of fruit detection using such a technique has been shown to have equal accuracy in broad daylight and nighttime. Besides, it will also protect natural elements, such as small wind and precipitation sprints.

Zhang used a color camera with a ring flash for capturing the images at night time, examining the characteristics of green apples and near-color backgrounds, a classifier was intended to accomplish a green background dependent on the shading attributes.

2.2 Segmentation and Feature Extraction

Segmentation is also considered in computer vision systems as a significant cause of error [11]. It is a sensitive stage prone to image quality, particularly variation in the intensity of incidents over the image that can change the segmentation aspects dramatically. The use of the color difference between fruits and the unwelcome sense is an obvious and practiced approach often addressed in this study [11] [5]. These attempts are primarily successful in detecting fruits of varying colors relative to the background.

Artificial Neural Network was used by Kurtulmus et al. [12] for apple classification. By extracting the features from peach canopy images and constituting distinct image scanning techniques, color, shape, and texture information was used. Computer vision algorithms were created using color images taken from a natural canopy to detect and count immature peach fruits under natural illumination conditions. Linker et al. [8] and Cohen et al. [13] used KNN clustering for apple classification. Qiang et al. [14] In this research, a machine vision system was built to accomplish thesetasks, consisting of a color CCD camera and a computer. Photos of citrus trees in sunny and gloomy conditions were captured. The red, green, and blue values of objects in theseimages are drastically altered due to varying levels of lightness and location randomnessof fruits and branches.

Deep Learning for image segmentation and objection recognition is the newest beginning of-of-the-art of machine vision. Deep learning for image segmentation based on the Convolutional Neural Network (CNN), such as Fully Connected CNN [15]. Recurrent Neural Networks (CRF-RNN) [16] Positive results in pixel classification have been demonstrated by Conditional Random Fields. These strategies make a progression of attributes that make them a solid visual system and give end-to-end learning abilities. The primary advantage of these organizations is the capacity to tweak the pre-prepared network parameters to adjust to custom testing tests. computer vision applications in farming may profit altogether from such techniques for segmentation.

Fruit identification and localization are other important steps after segmentation in the vision process. This step involves finding the spatial orientation as well as the three-dimensional location of each fruit in the photographs and canopies. For the detection of fruits, shape features specific to fruits are also used in segmented images.

Single Shot MultiBox Detector (SSD) [17] was used by Yuki et al. for the detecting and

To categorize the fruit's two-dimensional (2D) position. One of the general object recognition techniques used by the SSD is Convolution Neural Network (CNN). The SSD will comprehensively judge from color and type. A three-dimensional (3D) location needsto be acquired to submit a command to the robot arm. A stereo camera is used to measurethe 3D location of the fruit observed by the SSD.

2.3 Conclusion for the literature review

- Despite this, these techniques are difficult to use in varying light conditions because color information cannot be obtained quickly enough. To address challenges including clustering and variable light conditions, Multiple features such as color, shape, texture, and reflection can be used to detect and locate fruit.
- Without adequate thinning and pruning of planner tree canopies, such as tall spindle trees in vertical trellis or v-trellis architectures, the task of fruit identification remains difficult.

- Although cutting-edge research continues to detect fruit in its natural state, Artificial Intelligence (AI) for object detection continues to improve and become more effective.
- With practical methods, the vision system's harvesting efficiency could be improved. An iterative approach to imaging and harvesting the most visible fruits may be one such technique. This method is independent of the apple identification process and may be used to minimize canopy complexity strategically.
- The visibility of occluded and clustered fruit would increase with subsequent imaging and harvesting cycles, making total fruit detection more accurate [18]. A simple solution has already demonstrated that fruit identification accuracy of 98 percent can be achieved using this method.
- There is currently much less domestic and international literature on night photographs for agriculture. However, thanks to the contributions of several academics, some progress has been achieved.
- In terms of construction, and only minor change has been made. of a commercially viable apple harvesting system using robotic technology. Commercial performance has been limited due to the extremely unstructured and complicated canopy architectures, unpredictable outdoor environments, size and form inconsistency, and fragile existence of fruit.

2.4 Objectives of the thesis

- Gathering a deep understanding of the equipment’s needs to be used in this project.
- Choose the most advanced method for identification and harvesting.
- Create an algorithm for the identification and localization of apple using a camera.
- Create and train the image dataset and use it as an input for a neural network that is similar to a CNN and predicts the classes and detects the object.
- Improve the accuracy of the method of apple detection and make the harvesting time up to 5sec.
- Develop and improve its harvesting ability.

3. METHODOLOGY

The emphasis of this chapter will be on a thorough overview of how the desired solution will be implemented. The outline of the suggested solution will be explained in the first part of the chapter.

The hardware and the software platforms that were chosen, also the communication parameters that were used in the project will be explained in the next chapters.

The final section will conclude with a comparison of the effectiveness of the alternative methods that were evaluated before the desired method was chosen.

3.1. Object detection

Object Detection is the problem of locating and classifying objects in an image. which is a computer vision technique that allows a software device to detect, locate, and track an object in an image or video. Object detection distinguishes itself by identifying the type of object (person, car, mobile, etc.) and its location-specific coordinates in the picture. A bounding box is drawn around the item to indicate its position. It’s more than just a classification, in a classification process there is only one object in each image, and these can predict only one label. But in object detection, each image may contain multiple objects. Hence the model classifies objects and identifies their location.

Object detection is sometimes also known as object recognition, localization there are multiple phases. it is good to differentiate detection versus recognition versus localization. Localization of an object means finding out the area in the image where the object lies, localizing the object inside the image, or finding out its bounding box coming up with the boundary of the object. Detection sometimes is referred to as finding out whether there exists a particular object in the image or not. so, for example, this particular object that exists in this image so is called detection the object is detected. Then object recognition or sometimes called classification means to find out the category of that object what is the identity of that object.

When in computer vision literature in image processing literature when people talk about object detection one way or the other and in most of the cases, they're actually referring to all these three phases in which the object is detected it is recognized or classified, and also it is localized. Although the three problems are complicated when they come up in combination when normally people talk about object detection in most of the cases referring to the three phases and detect the object whether there is an object in the image or not if there is an object in the image what is the object and where the object is in the image all these three faces sometimes is referred to as object detection although a better way is to represent detection as well as localization as well as recognition so maybe differentiating the terms differently might be great.

When it comes to object election, really means mostly detection recognition and localization and it turns out this problem. In this kind of three combinations is not easy it's hard when it comes to real images because one image can contain a lot of objects the objects may be overlapping may be occluded by the other maybe only parts of the objects are visible the objects may appear in different sizes different rotations in anywhere in the images. There are a lot of challenges in finding out all the available objects in the image when the objects have a lot of variations and there is background clutter as well.

Object identification can be done using one of two methods:

I. Algorithms based on classification (Two-shot detection)

II. Algorithms based on regression (Single-shot detection)

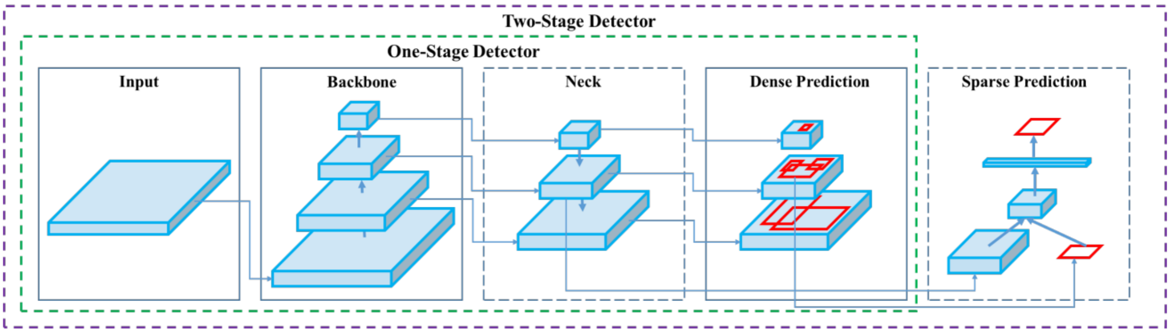


Figure 3. 1 object detector [19]

Input: This is where to input image { images, patches,… }

Backbone: This is the network that takes the image as input and extracts the function map {VGG 16 CSPResNeXt50, CSPDarknet53, and EfficientNet B3…..}

Neck: The Neck and head are sub-sets of the backbone, that help to improve function discriminability and robustness using FPN, PANet, Bi-FPN, and other techniques.

Head: which is in care of the prediction This can be done with a one-stage detector like Yolo or SSD for dense prediction, or a two-stage detector like Sparse Prediction.

Dense prediction :{RPN, YOLO, SSD, RetinaNet, FCOS,…..}

Sparse Prediction: {Faster R-CNN, R-FCN,…}

3.1.1 Naive Approach

Convolutional Nets as great as they are at classification also help at detection. Taking an existing convolutional net and repurpose it as an object detector, consider any existing classifier and use it for the detection of a particular image. It was trained on core images amongst many other classes. For example, figure 3.2 contains two objects and also assumes that have a set of predefined classes. here class 1 refers to a dog and class 2 refers to a person and provided a large image. So, the process to find out whether this particular batch is a particular object under these objects. Consider the model has a lot of training data in which it has a lot of same batches size images and have some classifier that is trained on these fixed-sized images a lot of these and assume using some feature extractor may be HOG or LBP or several feature extractors. Extract the features and each image will be represented by a vector of numbers and then predict a label for that location, by feed the images into a classifier model. classifier classifiers that particular image is of what kind of category either it’s a dog or person or none of these. once the object category or the class is found then localization is can be carried out.

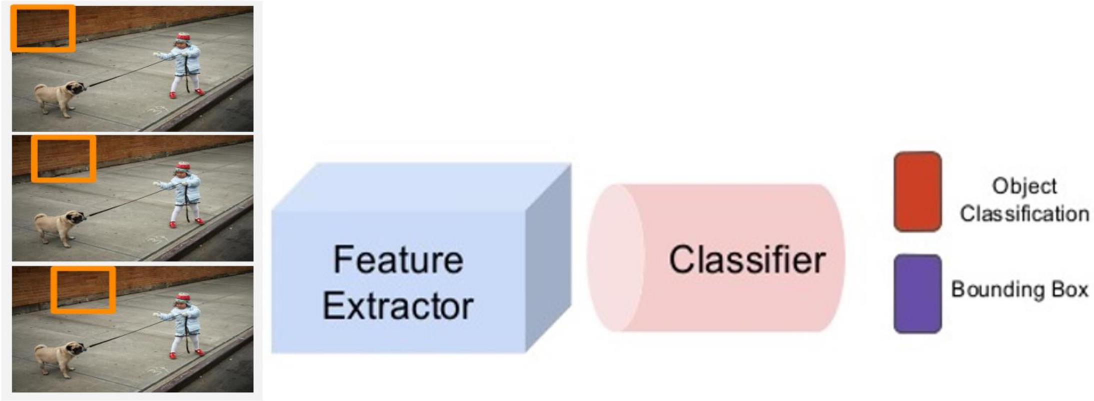


Figure 3. 2 object detection pipeline [20]

This image is larger than have to search for the same batch everywhere in the particular image sometimes known as the sliding window. Scan the image with a sliding window classifier over every single bunch of squares of that image right so it can classify every single region as defined, Get a lot of different classifications from there.

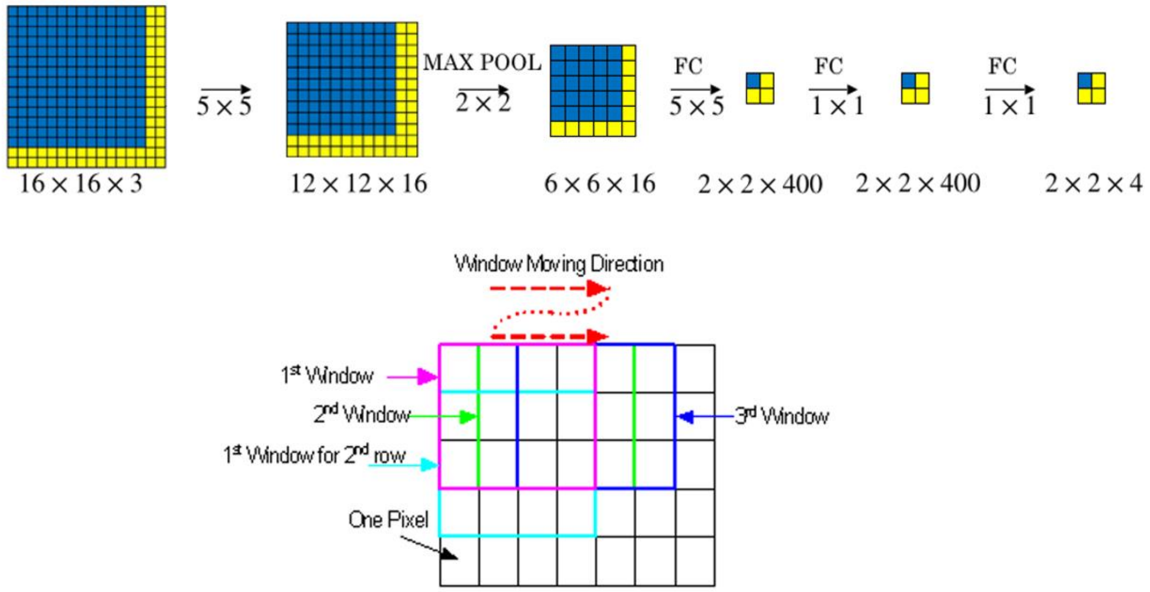


Figure 3. 3 sliding window [21]

Then only keep the ones that the classifier is the most certain about and use that to bend draw a bounding box around the image but this is a very computationally expensive approach. This method is slow since it tests several windows that are empty. It is not suitable for real-time applications. That is not a positive approach. A better solution is needed and there must be a more reliable way of doing it correctly. The Region-based Convolutional Neural Net (R-CNN) is an improved version that strategically selects regions to run through the CNN that are likely to contain an entity.

3.1.2 Algorithms based on classification (Two-shot detection)

Algorithms based on classification or form of detection with two shots, This process is divided into two stages, as the name implies. The first step is the region proposal, followed by the classification of certain regions and refining of the position prediction in the second stage. First, we use the picture to choose regions of interest (ROIs). Then, using convolutional neural networks, identify those regions.

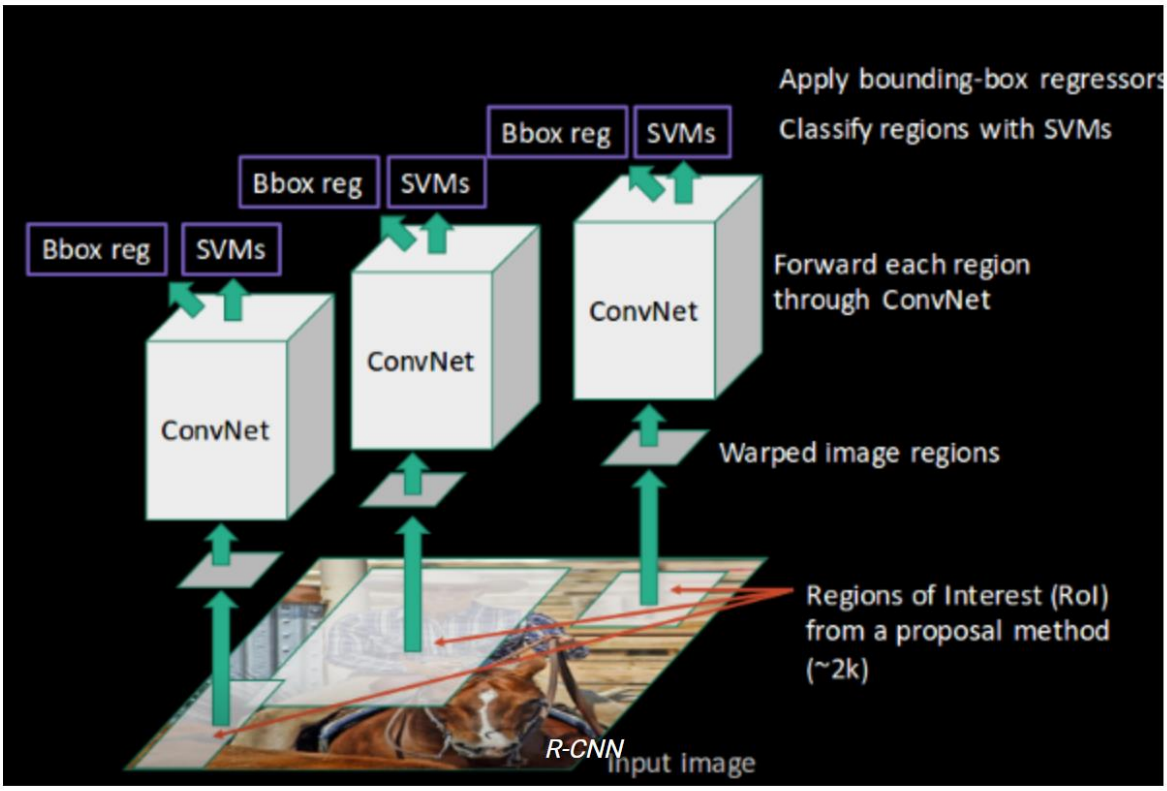

Apply bounding-box regressors
Classify regions with SVMs
Bbox reg
SVMs
Bbox reg
SVMs
Bbox reg
SVMs
ConvNet
ConvNet
ConvNet
Forward each region through ConvNet
Warped image regions
Regions of Interest (RoI) from a proposal method (~2k)
R-CNN
Input image

Figure 3. 4 R-CNN algorithm [22]

R-CNN creates bounding boxes, region proposals, using a process called selective search. At a high level, the selective search process looks at an image through a series of windows of different sizes. These are just randomly size placed windows and for each size, it tries to group adjacent pixels by texture color or intensity to identify objects. Given some input image but the first step is to generate a set of region proposals bounding boxes for however want by some threshold. Once it’s done, run those images in the bounding boxes through a pre-trained CNN to compute the features for that bounding box and then finally a support vector machine (SVM) that classified to see what the object the image in the box. run the box to a linear regression model to output tighter coordinates for the box once the object has been classified.

Algorithm	**Features**	**Prediction time**	**Limitations**
CNN	Divides the image into several regions, which are then classified into different classes.	-----	To predict accurately, a large number of regions are needed, resulting in a long computation period.
R-CNN	To produce regions, it employs selective search. From each image, about 2000 regions are extracted.	40-50 seconds	Since each region is passed to the CNN separately, there is a long computation period. It also makes predictions using three different models.
Fast R-CNN	The CNN receives each image only once, and feature maps are extracted. On these maps, selective search is used to produce predictions. All three R-CNN models are combined in this feature.	2 seconds	Since selective search is slow, computation time remains high.
Faster R-CNN	The area proposal network (RPN) replaces the selective search process, making the algorithm much faster.	0.2 seconds	Object proposal takes time, and since several systems are operating one by one, each system's output is influenced by the previous system's performance.

Table 3. 1comparison of RCNN models

This proved to be an effective approach for object detection. The most effective by repurposing a convolutional network for object detection by using this selective search algorithm to create bounding boxes, beforehand and then feeding all those boxes to a CNN to compute a list of features and then, computing class values from them. For R-CNN there have been lots of improvements like Fast R-CNN, Faster R-CNN, and Mask R-CNN. Due to the various steps involved in the process, R-CNN becomes very slow.

3.1.3 Algorithms based on regression (Single-shot detection)

While two-shot detection models perform better, single-shot detection is in the sweet spot of performance and speed/resources, making it more suitable for tasks such as detecting objects in live feed or object tracking where prediction speed is more important. This can predict classes and bounding boxes for the image in one run of the algorithm rather than choosing a region of interest (ROIs) from the image. YOLO (You only look once) is an example of this type of algorithm. Single-shot detection is used to identify the apple and PointNet to process the point clouds implemented in Hanwen et.al's [23] apple harvesting robot.

As previously mentioned, YOLO stands for "You Just Look Once," and it is a single shot detection algorithm developed by Joseph Redmon in May 2016. Although the algorithm's name accurately describes the algorithm since it predicts classes and bounding boxes for the entire image in a single run. As compared to other single-shot detectors at the time, YOLO outperformed them in terms of speed and accuracy. It is not the most accurate algorithm for object detection, but it more than makes up for it with its impressive pace, making it a great combination of speed and accuracy. Rather than making predictions on multiple regions of an image, YOLO sends the entire image to a CNN, which predicts labels, bounding boxes, and confidence probabilities for entities in the image, as seen in the diagram below.

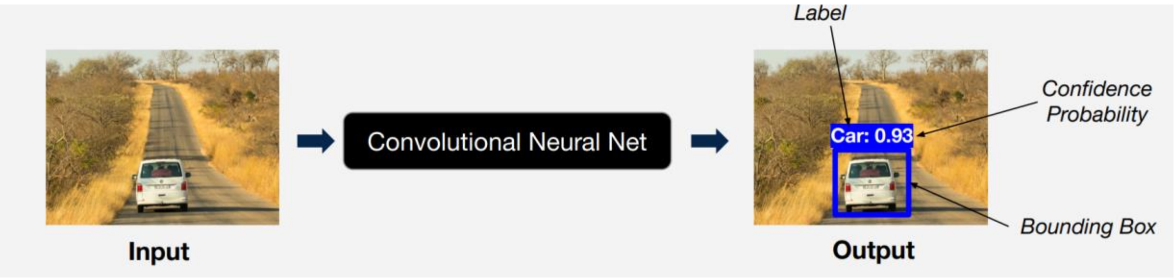


Figure 3. 5 YOLO algorithm basic structure [20]

YOLO is a state-of-the-art object detection algorithm that is incredibly fast and accurate. Yolo is the best approach because it outperformed CNN and all its variants. Yolo takes a completely different approach. YOLO achieved state-of-the-art mAP(mean Average Precision) at the time of first publication (2016) as compared to networks like R-CNN and FRCNN . There are some speed and accuracy improvements in the newer edition. As of January 2021, there is a total of 5 versions of YOLO.

1. YOLOv1 [24]
2. YOLOv2 and YOLO9000 [25]
3. YOLOv3 [26]
4. YOLOv4: Optimal Speed and Accuracy of Object Detection [19]

3.1.4 YOLOv4

In 2016 the first version of yolo was invented by Joseph Redmond. in 2017 they came up with yolo version 2, Where they improved the speed which supports the rate of 67 frames per second. so, this algorithm detects the objects at a very fast rate. In 2018 YOLOv3 was created and again Joseph and Ali were the pioneers in this incremental improvement. so the captain improving the original Yolo framework till version 3 and then at some point Joseph Redmond stops working on it because of some concern. From there in 2020 the YOLO v4 was created by Alexey Bochkovskiy, he created a fork of his GitHub repository and he created this separate branch of this version for algorithm now nowadays there is Yolo v5 e as well but there is some controversy going on with it.

In this project, YOLOv4 will be chosen to build the model and use pre-trained weights to detect boundaries in an image. The original model was trained on the COCO dataset, which contains 1.5 million objects of 80. According to the authors, YOLOv4 aimed to develop a fast object detector for production systems that was also optimized for parallel computations. It needed to be lightning-fast, accurate, and produce compelling object detection results. From figure 3.6 it is clear that YOLOv4 can reach 43.5% AP on COCO and the speed can reach 65 FPS.

What changed in YOLOv4:

- **Bag-of-Freebies** - Specific changes to the training method that increases accuracy while not affect inference speed (e.g., data augmentation, class imbalance, cost feature, soft labeling, and so on);
- **Bag-Of-Specials**- is a network improvement that has a minor impact on inference time but a good return performance in terms of results. Increased receptive field, attention, feature integration like skip-connections, and post-processing like non-maximum suppression are all examples of these enhancements.

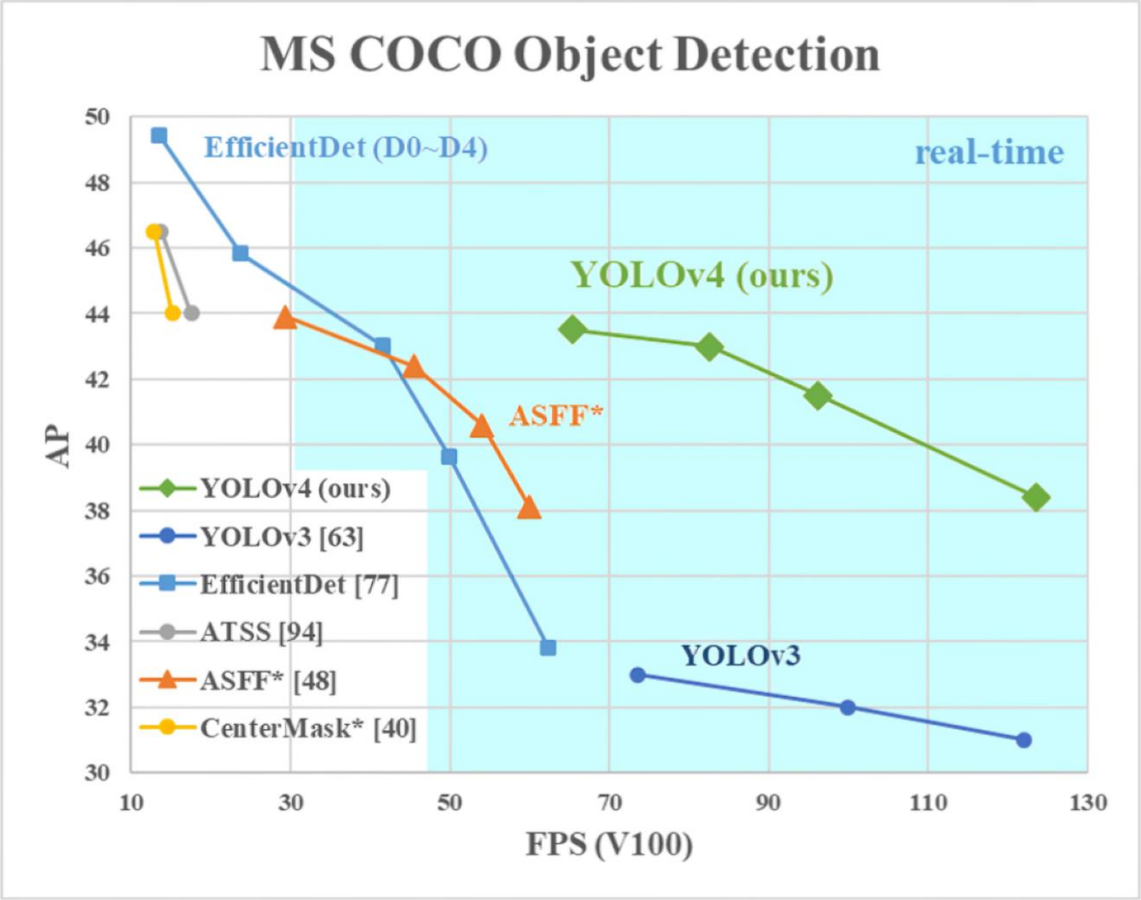
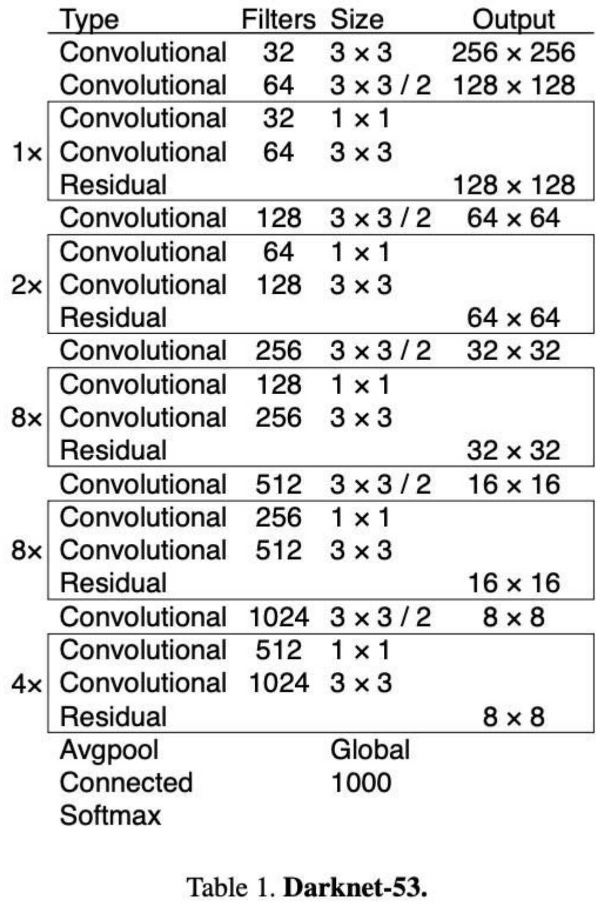


Figure 3. 6 YOLOv4 results on MS COCO Object detection [19]

In terms of both speed and precision, it outperforms the fastest and most reliable detectors. With comparable efficiency, YOLOv4 is twice as fast as EfficientDet. YoloV4 is a significant upgrade over YoloV3. The introduction of new architecture in the Backbone, as well as changes in the Neck, have increased the mAP (mean Average Precision) by 10% and the number of FPS (frames per second) by 12%. Furthermore, training this neural network on a single GPU has become simpler.

Object detectors' backbone networks are usually pre-trained using ImageNet classification every object detector starts with an image and extracts features using a convolutional neural network backbone. Multiple bounding boxes must be constructed around images for object detection, as well as classification, therefore the feature layers of the convolutional backbone must be blended and held up in the sight of one another. The neck is where the backbone feature layers come together.

In feature extraction, YOLOv4 uses the CSP connections with the Darknet-53 below as the backbone.



	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			

Table 1. **Darknet-53.**

Figure 3. 7 darknet 53 architecture [27]

Even while ResNet-based designs have greater classification performance, the CSPDarknet53 model has higher accuracy in object detection. However, using Mish and other techniques, the classification accuracy of CSPDarknet53 can be enhanced.

The final architecture is like;

- Backbone – CSPDarknet 53
- Neck – SPP + PANet
- Head – YOLOv3

YOLOv4 employing a new data augmentation technique and an improved object detection network architecture. To make their YOLOv4 design suitable for efficient training and detection, the authors used and merged the following new features:

- Weighted-Residual-Connections (WRC)
- Cross-Stage-Partial-Connections (CSP), A new backbone that can enhance the learning capability of CNN
- Cross mini-Batch Normalization (CmBN), represents a CBN modified version which assumes a batch contains four mini-batches
- Self-adversarial training (SAT), represents a new data augmentation technique that operates in 2 forward-backward stages
- Mish-activation, A novel self-regularized non-monotonic neural activation function
- Mosaic data augmentation, represents a new data augmentation method that mixes 4 training images instead of a single image
- DropBlock regularization, a better regularization method for CNN
- CIoU loss achieves better convergence speed and accuracy on the BBox regression problem.

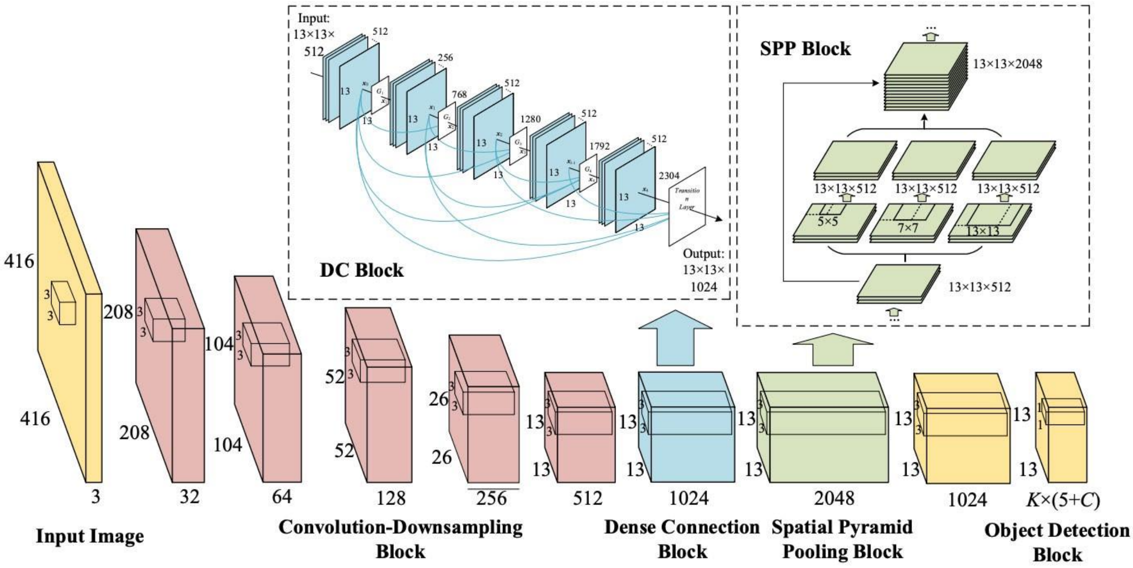


Figure 3. 8 Network structure [28]

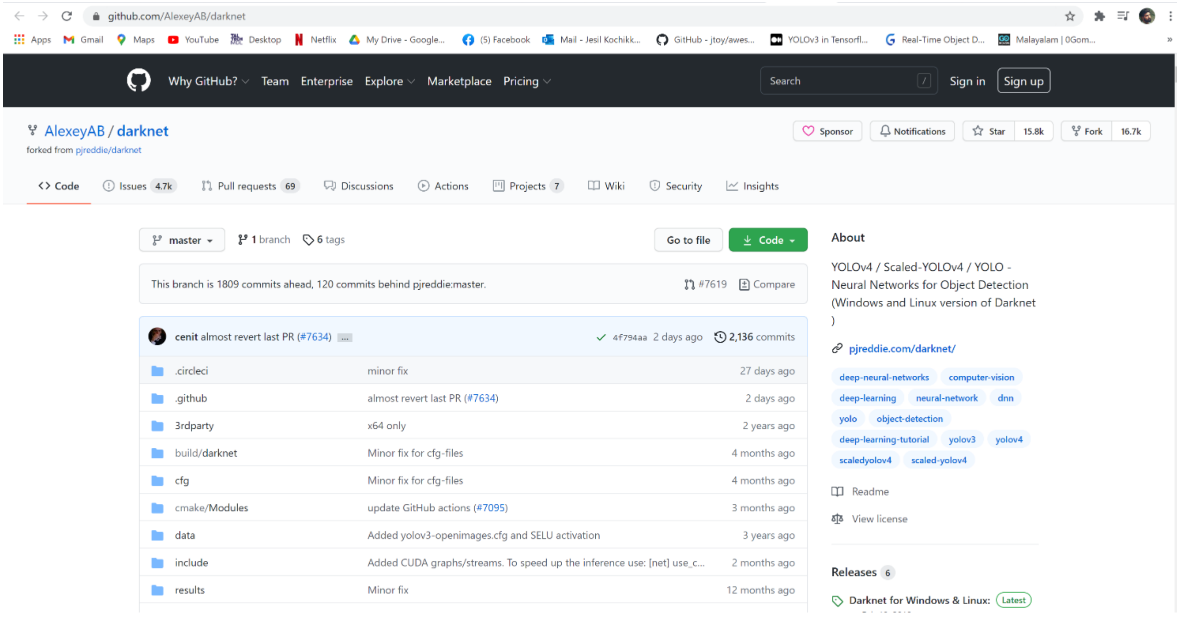
For the detection of fruits such as oranges, apples, and mangoes. Wan et.al [29] made a comparison with standard Faster R-CNN and YOLOv3. . On the same dataset, the modification proposed in this paper uncovers about 90% of the fruits, which is 3–4% better than the normal Faster R-CNN and on equal ground with YOLOv3. However, YOLOv3 had a 40-millisecond average recognition time, compared to 58 milliseconds for the updated Faster R-CNN network and 240 milliseconds for the normal Faster R-CNN network. Wenkang et.al [30] used the combination of the Canopy algorithm and the K-Means++ algorithm with YOLOv4 for the detection of citrus fruit in a natural environment.

YOLOv4 has the following advantages over its older version, YOLOv3;

I. Propose an efficient and powerful target detection model. It allows everyone to use 1080 Ti or 2080 Ti GPU to train ultra-fast and accurate target detectors.

II. During detector testing, the effect of state-of-the-art "Bag-of-Freebies" and "Bag-of-Specials" object detection methods has been validated.

III. State-of-the-art methods such as CBN (Cross-iteration batch normalization), PAN (Path aggregation network), and others have been improved to make them more effective and appropriate for single GPU training.



This branch is 1809 commits ahead, 120 commits behind pjreddie:master.

cenit almost revert last PR (#7634) ... 4f794aa 2 days ago 2,136 commits

.circleci	minor fix	27 days ago
.github	almost revert last PR (#7634)	2 days ago
3rdparty	x64 only	2 years ago
build/darknet	Minor fix for cfg-files	4 months ago
cfg	Minor fix for cfg-files	4 months ago
cmake/Modules	update GitHub actions (#7095)	3 months ago
data	Added yolov3-openimages.cfg and SELU activation	3 years ago
include	Added CUDA graphs/streams. To speed up the inference use: [net] use_c...	2 months ago
results	Minor fix	12 months ago

YOLOv4 / Scaled-YOLOv4 / YOLO - Neural Networks for Object Detection (Windows and Linux version of Darknet)

pjreddie.com/darknet/

deep-neural-networks computer-vision deep-learning neural-network dnn yolo object-detection deep-learning-tutorial yolov3 yolov4 scaledyolov4 scaled-yolov4

Darknet for Windows & Linux:

Figure 3. 9 Darknet GitHub repository [31]

Figure 3.6 shows the official website of YOLO GitHub has been updated by AlexyAB and officially added YOLOv4's papers and code links. which is the original repository by Joseph Redmond and he stopped the development of Yolo after version 3 so then AlexyAB forked his repository and he builds this yolo version 4. In short, with YOLOv4, we are using a better object detection network architecture and new data augmentation. techniques.

3.2 Hardware Platform

This hardware section gives a detailed overview of the devices used in this project. The important hardware used in the project for capturing images and testing the model is Intel® RealSense™ Depth camera D415 and Logitech HD C615 Webcam. The specifications of the devices are discussed below.

3.2.1 Experimental Platform

The apple vision system model implemented and tested on computer with the following specification.

Technical requirements	**Specifications**
processor	Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz 2.90 GHz
Installed RAM	16.0 GB (15.7 GB usable)
System type	64-bit operating system, x64-based processor
Display adaptors	Intel(R) UHD Graphics 630 CPU
GPU Name	TU104
GPU	NVIDIA GeForce RTX 2080 SUPER 8GB
Windows specification	Windows 10

Table 3. 2 Hardware specification

GPU	Stream processors	Texture units	Tensor cores	RT cores	Boost clock	Video memory	Memory speed	Power connections
NVIDIA GeForce RTX 2080 SUPER	3072	192	384	48	1815 MHz	8GB DDR6	15.5 Gbps	1 x 8-pin; 1 x 6-pin

Table 3. 3 GPU specification

The following prerequisite needed to compile darknet YOLOv4 in a machine with OpenCV and GPU acceleration.

- Anaconda with Python 3.8
- Visual studio 2019 community version
- CUDA v10.1
- CuDNN v7.6.5
- OpenCV v 4.5.2
- Cmake 3.20.0
- Pycharm community edition 2021.1.1

3.2.2 Intel® RealSense™ Depth camera D415

The vision system required a high-resolution RGB camera to capture good-quality images. The camera should provide almost every detail of the object. The Intel® RealSense™ D415 has a standard field of view well suited for high accuracy applications such as 3D scanning. The D415 is an excellent alternative when precision is critical to the solution. The D415 has a narrow field of view that provides better resolution depth per degree. The D415 is ideal for facial recognition, 3D scanning, and volumetric capturing thanks to its embedded RGB sensor.

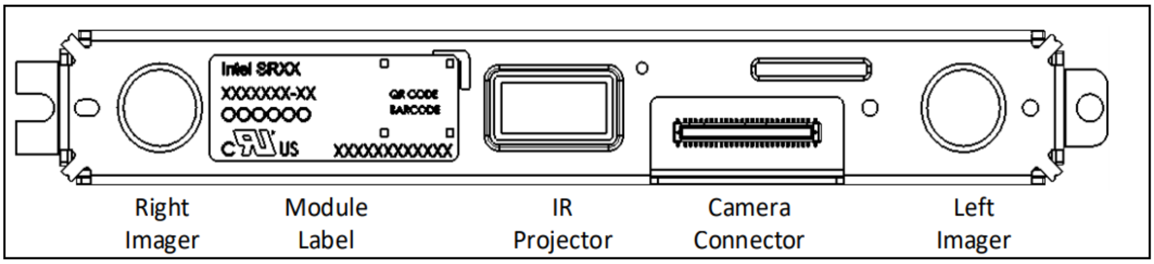


Figure 3. 10 Intel RealSense D415 camera and sensor details *[32]*

This camera puts an Intel module and vision processor into a small form factor ideal for the development of a vision system. It has features like Lightweight, powerful, and low-cost, it pairs with customizable software suitable to access. Figure 3.5 shows the actual view of the camera. It is compact with dimensions of 99mm x 20mm x 23mm. The bottom image shows the included hardware camera modules. The camera can be connected to the PC using a USB 3.0 connection and the camera has a USB Type C interface.

Device	**Use environment**	**Ideal range**	**RGB sensor technology**	**RGB frame resolution**	**RGB sensor FOV (H × V × D)**	**RGB frame rate**	**RGB sensor resolution**
Intel RealSense D415	Indoor / Outdoor	0.5 m to 3 m	Rolling Shutter, 1.4µm× 1.4µm pixel size	1920 × 1080	64° × 41° × 77 ° (±3°)	30 fps	2 MP

Table 3. 4Intel RealSense D415 camera specification

3.2.3 Logitech HD Webcam C615

Logitech C615 Webcam is used to test the trained model capture images and test the image to detect the object classification. For harvesting robots, the best operational distance between the camera and apple canopy trees in the field was 0.4–1.0 m. C165 provides fast autofocus that stays razor-sharp even if the object is moving also, equipped with HD light correction. Specifications of the webcam as follows.

Figure 3. 11 logitech C615 HD webcam

Camera type	**Sensor resolution**	**Frame resolution**	**Frame rate**	**Connectivity**
webcam	8 Megapixel	1920 x 1080 HD	30 fps	USB 2.0

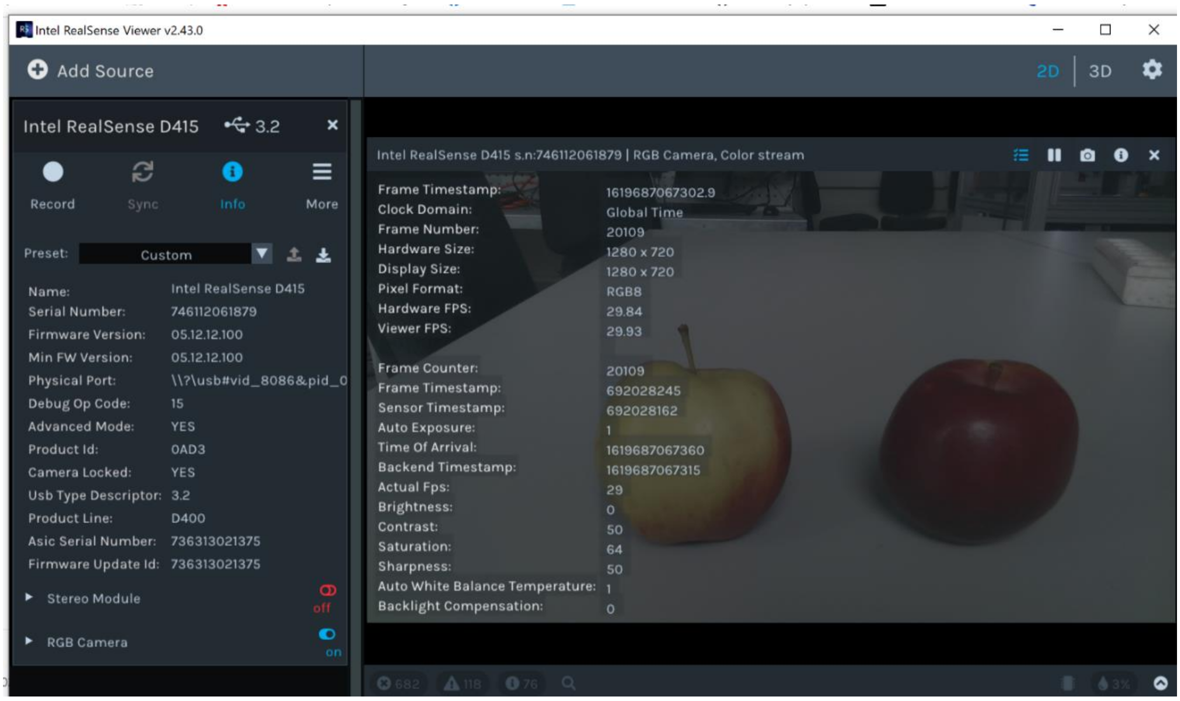
Table 3. 5 DELL XPS 13 9360 Webcam specification

3.3. Software Platform

This section gives a detailed overview of the required software to use in this project

3.3.1 Intel® RealSense™ Software Development Kit 2.0

The Intel® RealSense™ Software Development Kit 2.0 (SDK 2.0), is a cross-platform operating system independent software that connects the PC with the camera.



Preset: Custom

Name:	Intel RealSense D415
Serial Number:	746112061879
Firmware Version:	05.12.12.100
Min FW Version:	05.12.12.100
Physical Port:	\\?\usb#vid_8086&pid_0
Debug Op Code:	15
Advanced Mode:	YES
Product Id:	0AD3
Camera Locked:	YES
Usb Type Descriptor:	3.2
Product Line:	D400
Asic Serial Number:	736313021375
Firmware Update Id:	736313021375

Intel RealSense D415 s.n:746112061879 | RGB Camera, Color stream

Frame Timestamp:	1619687067302.9
Clock Domain:	Global Time
Frame Number:	20109
Hardware Size:	1280 x 720
Display Size:	1280 x 720
Pixel Format:	RGB8
Hardware FPS:	29.84
Viewer FPS:	29.93
Frame Counter:	20109
Frame Timestamp:	692028245
Sensor Timestamp:	692028162
Auto Exposure:	1
Time Of Arrival:	1619687067360
Backend Timestamp:	1619687067315
Actual Fps:	29
Brightness:	0
Contrast:	50
Saturation:	64
Sharpness:	50
Auto White Balance Temperature:	1
Backlight Compensation:	0

Figure 3. 12 RealSense SDK2.0 interface

RealSense SDK2.0 also known as librealsense, is a cross-platform and open-source software development kit from Intel. Ubuntu* 16.04 and Windows® 10 are both sponsored by the SDK. The SDK includes tools, code, samples, and wrappers for a variety of languages for extracting data from Intel® RealSenseTM Depth cameras. The D415, D435, and SR300 cameras are supported by Intel® RealSenseTM SDK 2.0.

3.3.2 Darknet

Darknet is an open-source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation. Darknet is a Deep learning library created by the author of the YOLO paper, and the original implementation (made by the author of the paper) is provided. Darknet is primarily used for Object Detection, and its design and functionality vary from those of other deep learning systems. It outperforms many other NN architectures and approaches, such as Faster RCNN and others. YOLO is a specialized platform for darknet architecture, and they are at the top of their game in terms of speed and accuracy. YOLO can run on a CPU, but it runs 500 times faster on a GPU because it uses CUDA and cuDNN.

This deep learning framework is written in C, but once the network has been trained, Darknet is no longer needed for inference. Darknet formats are natively supported by OpenCV, so both model and qualified weights can be used anywhere OpenCV is installed, including from Python.

On the plus side, this network comes with some standard documentation on how to train your own data set and run inference on your own data. Other common frameworks are often so heavily "optimized" for training and validation against various existing data sets that breaking out of this golden cage and building a functional product becomes shockingly difficult. Since the trained model is distributed, it is easy if you just move it

3.3.3. Google Colab Notebook

Google Colab is a free Google service that allows running Python scripts and machine learning libraries on Google's strong hardware. Google colab has been used for the computational process in this project. Colab is a GPU-based Python Jupyter notebook and it is free to use, but can be upgraded to a Pro account by payment**.**

Because it’s free to use, it also has some minor issues as follows;

- When attempting to connect to GPU runtime, an error message appears stating that it is unable to connect. This is attributable to a high volume of users attempting to use the service. Google, on the other hand, intends to add more GPU machines.

- The usable RAM is 13GB, and the disk space is 64GB which is much too fine to be giving away for free. However, in the case of object detection, computer vision, and machine learning, could run into memory warnings with broad models.

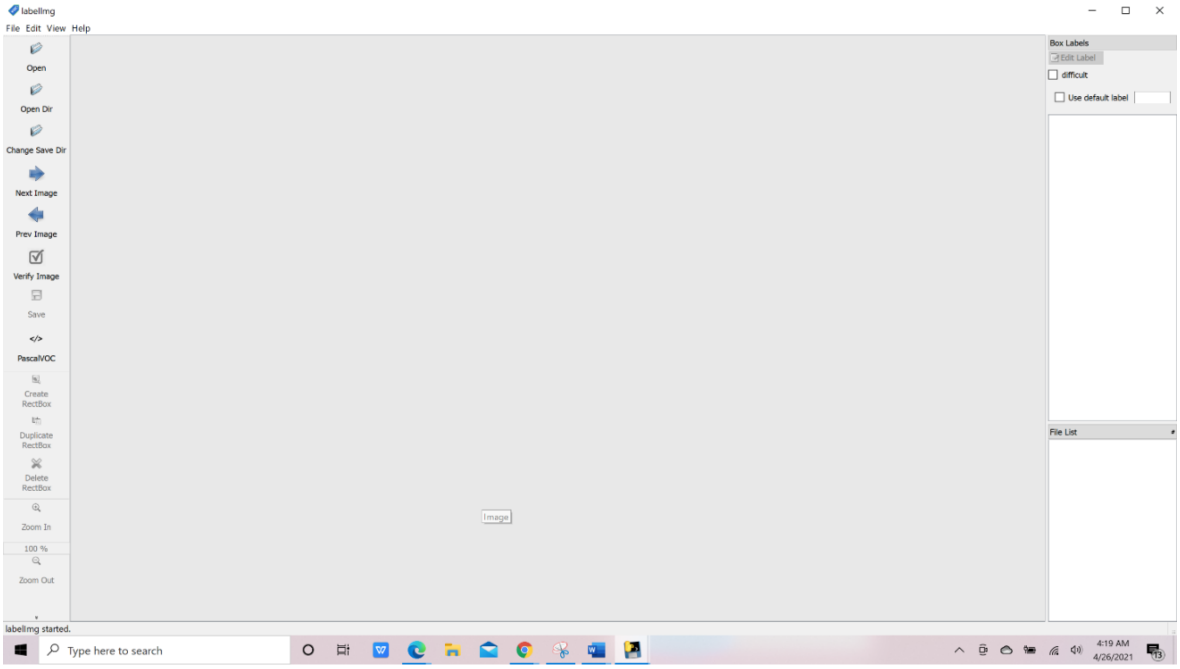
- The runtime simply dies from time to time. There may be a variety of reasons for this.

- It is online, with the drawback that can only use it for 12 hours in a row before being disconnected and our files being erased. we can restart it, though we have to start again from the beginning.

- Overall, if we want to train a big model, we can run into memory and space constraints. We will fix the 12-hour issue and file loss problem by linking Google Colab to Google Drive, ensuring that we don't delete data as a result of a disconnect.

3.3.4 LabelImg

Images with bounding boxes must be labeled in order to train a YOLO model. It can take some time to label these datasets. LabelImg is a free, open-source application for labeling images graphically. It's written in Python and has a graphical user interface built with QT. It's a simple and cost-free method of labeling. For this project downloaded the latest version of this software (App_vesrion: 1.8.0) because they used to keep on adding up functionality with this tool. The most difficult aspect of this procedure is making annotations. Both input images must be annotated with four coordinates indicating the position of an object's bounding box and its label.



labelImg started.

Figure 3. 13 LabelImg graphical user interface

This software contains multiple tabs on the left-hand side that can open a file from open command also can open up a complete directory. On the downside, see that there is a small button where it is possible to change the labeling types. There are three or four types over here for TensorFlow, Yolo, PyTorch likewise.

The following should be considered before the annotation process.

- Create a bounding box over the entire image do not skip any part of it. This way the model can easily understand the object.
- Occlusion is a major issue in this project. If an object is occluded then try to label

them entirely. Keep the objects inside the frame.

- The file name should be the same after annotation.
- Make sure that the file and annotation file has same name.

3.3.5 Visual studio 2019

Visual Studio 2019 is a development environment by Microsoft™ and it was used in the project to program to build the darknet. Visual studio used to include the OpenCV and change the project directories.

3.3.6 Pycharm 2021.1.1

PyCharm is a Python Integrated Development Environment (IDE) that includes a variety of key tools for Python developers that is unified to create a platform for web and data science development. It was used to deploy the darknet weight file to test the model.

.

4. DEVELOPMENT OF OBJECT DETECTION MODEL

This chapter will discuss in detail how object detection works in YOLO and how the model was developed. The first section of the chapter will provide the proposed solution for this project.

In the second section, how the YOLO algorithm works on single and multiple object detection. Give a detailed overview of intersection over union (IoU), Non-Max Suppression, anchor box, and mean average precision.

The third section will explain a concise flowchart that will include an outline of the framework. It's essential to provide a thorough description of the structure that was developed.

The fourth section will provide step-by-step information regarding training the YOLO on a custom dataset model. Image dataset preparation will be explained such as image acquisition, labeling, and annotation. setups google colaboratory notebook, importing files and enabling GPU so on.

In fifth and sixth sections explain the testing of the model in different methods. Testing should be over images captured by intel realSence camera and webcam, videos over real-time by the image capturing and live to stream.

4.1. Proposed Solution

Many deep learning models have been trained in recent years to detect various fruits on trees, including apples. However, the latest ‘harvesting robot prototypes' computer vision systems based on these models operate too slowly. The previous versions had many issues like do not detect apples with overlapping, green apples on a green background, occultation, and all. To solve this problem the author proposed to use the YOLO v4 algorithm, which is deployed in Darknet with a custom trained image dataset.

In this project, the author trains the Yolo “you only look once” version 4 object detector using Google Collab (Google's free cloud service for AI developers). The images of apples from different environments and lighting were used to train the model. The detector trained to check whether the apple is ripe or not. Once the training is finished the detector can run over images, videos, and live streaming. Since the detector needs to capture the image for processing.

The desired solution for object detection is described in the process flow pipeline figure 4.2 given below. In this project detection algorithm was based on color. Setting the color threshold makes it possible to determine which object belongs to which class. The image dataset contains 3 classes.

1.Ripe apple

2.Raw apple

3.Green apple

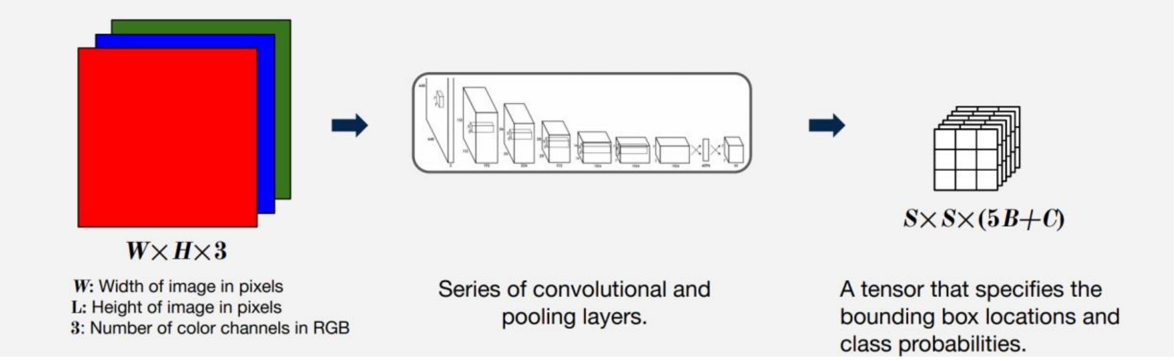

W×H×3
W: Width of image in pixels
L: Height of image in pixels
3: Number of color channels in RGB
Series of convolutional and pooling layers.
S×S×(5B+C)
A tensor that specifies the bounding box locations and class probabilities.

figure 4. 1 Basic object detection structure *[20]*

After labelling the dataset build the model using the darknet. Using transfer learning method to combine both configuration file and pre-trained convolutional weights to our model. These object detection algorithms might be pre-trained or can be trained from scratch. In most use cases, for this project author use pre-trained weights from pre-trained models and then fine-tune them as per the requirements and different use cases. Then bring the image as input, pass it through a neural network that is similar to a CNN, and train the custom model, it will take a while. Once the training process is finished, test the model using different approaches such as testing over image and video files than with image capture using camera and live streaming. In the output, there is a vector of bounding boxes and class predictions. Simply put an image as input, run it through a CNN-style neural network, and get a vector of bounding boxes and class predictions as output.

4.2 YOLO framework function

Object detection is handled differently in the YOLO system (You Only Look Once). It predicts the bounding box coordinates and class probabilities for these boxes using the entire image as a single case. Since only one pass through a CNN is needed, YOLO is much faster than region-based algorithms. The CNN forecasts the labels, bounding boxes, and trust probabilities for the image's artifacts.

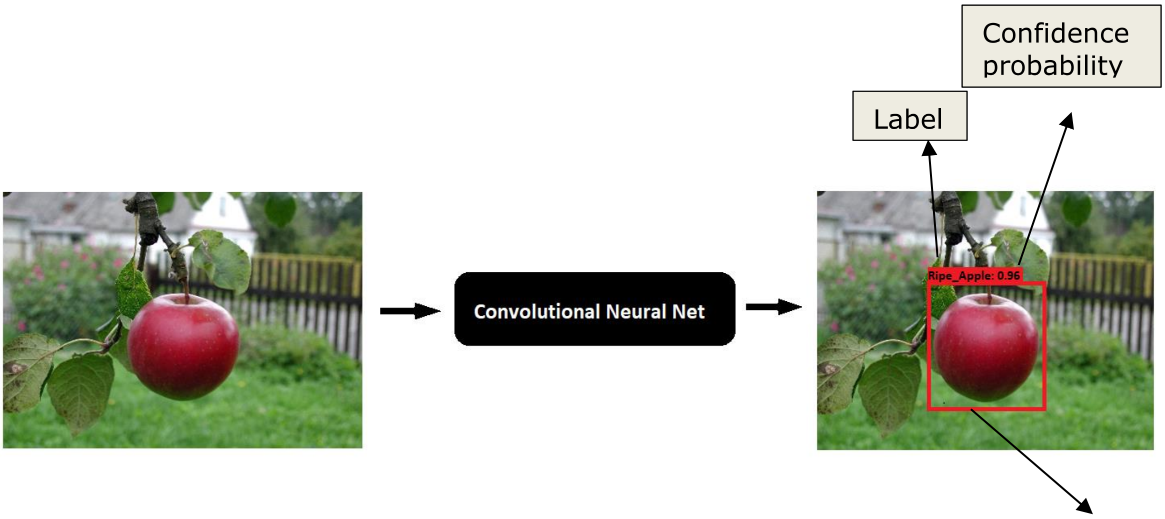


figure 4. 2 YOLO framework

So then the steps of the algorithm as follows;

YOLO first takes an input image, the framework then divides the image into cells with an S x S Grid.

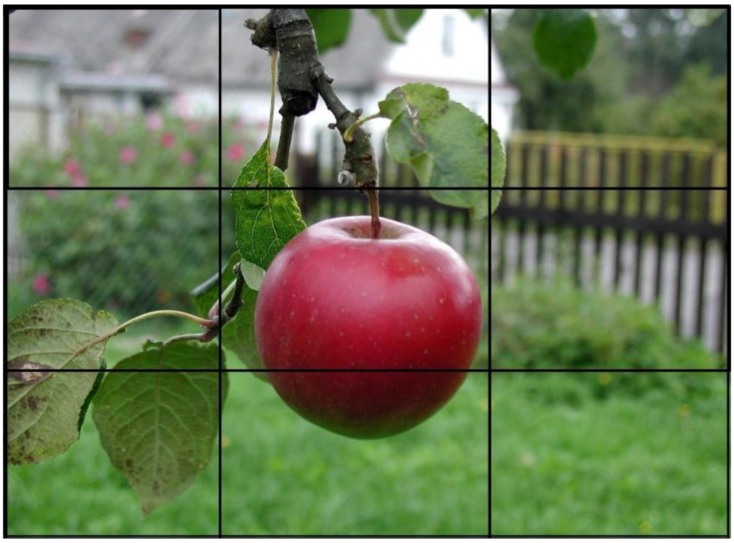


figure 4. 3 step 1 (S = 3)

In figure 4.3 each little box is a cell and S=3 so, divide the image into 3 x 3=9 cells here. Each cell is responsible for predicting the number of bounding boxes. The here is that S was a small number 3 but in practice use a larger number. It could be 13 x 13 or 19 x 19 there is no fixed rule that it has to be 3x3. That can predict way more objects in the image.

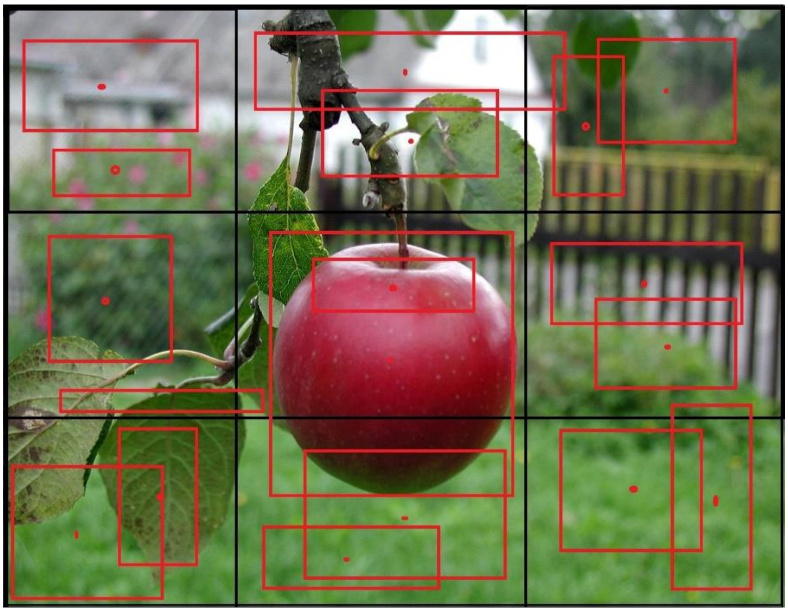


figure 4. 4 Step 2 (B=2)

Consider figure 4.4, where each cell predicts B bounding boxes. In this case B = 2. So, each cell here is filled with two boxes. And what means each cell is responsible for predicting two bounding boxes is that the center of the bounding box also within the cell and that cell should predict that bounding box. As shown in the picture, here is a new box around the apple. Sometimes these bounding boxes will extend beyond the cell and that's perfectly fine. As long as the center falls in the bounding box, that bounding box of the center the bounding box falls in the cell, that cell can predict the bounding box. That is a cell is responsible for detecting an object if the object's bounding box falls within the cell. In the figure, each cell has two red dots.

There is a lot of different bounding boxes in figure 4.4 that do not contain anything. there should have to set some confidence threshold, anything below the confidence interval is just gets removed and return when the boxes of the confidence interval as well. In below figure 4.5 shows, all these boxes are gone, and a box that has a higher confidence interval should go there, would be returned. And the rest of these will go somewhere else. The confidence probability of all other bounding boxes is less than the threshold, so they are suppressed.

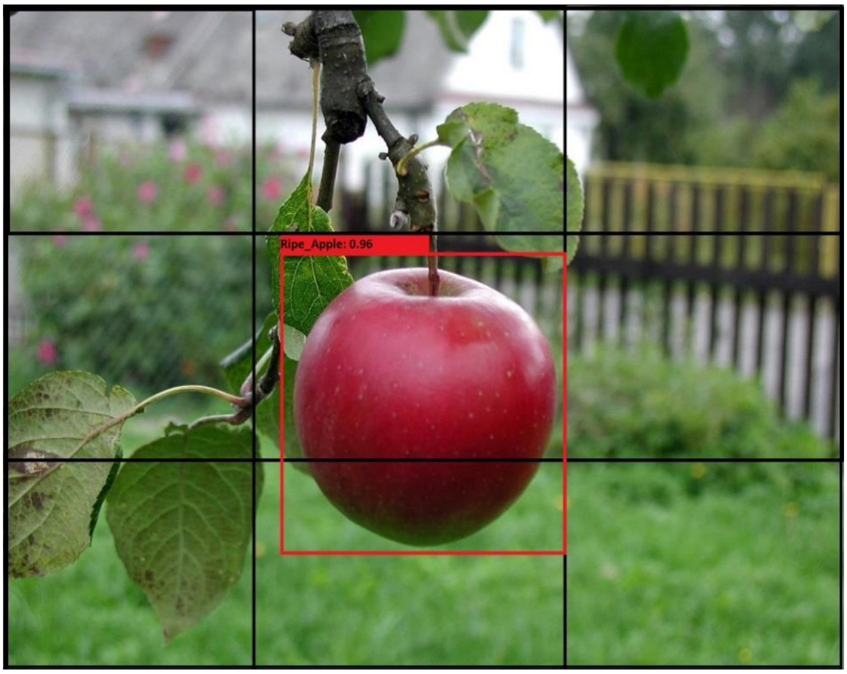
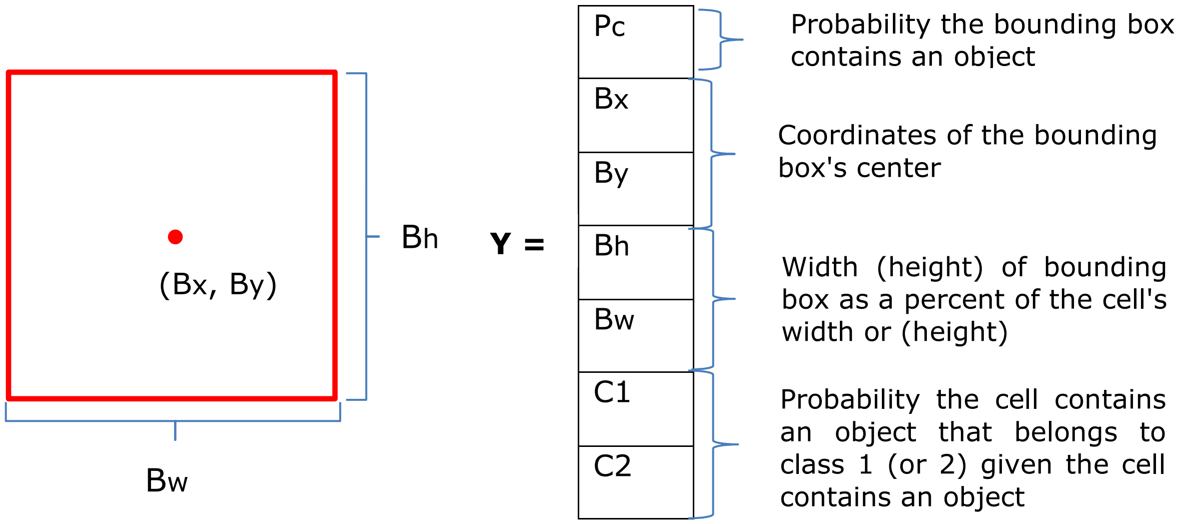


figure 4. 5 steps 3: Return bounding box with a higher threshold value

4.2.1 Encoding of Bounding Boxes

In object localization not only telling which class the object belongs but also telling the bounding box or the position of an object within the image. Consider figure 4.5, Consider a simple example in which there are 3x3 cells (S=3), each cell predicts 1 bounding box (B=1), and the objects are either ripe or green apples with class number 1 and 2 respectively. The CNN predicts a vector y for each cell.


(Bx, By)
Bh
Bw
Y =
Pc
Bx
By
Bh
Bw
C1
C2
Probability the bounding box contains an object
Coordinates of the bounding box's center
Width (height) of bounding box as a percent of the cell's width or (height)
Probability the cell contains an object that belongs to class 1 (or 2) given the cell contains an object

In figure 4.5, there is only one class ripe apple. So, the vector Y predicted by CNN will be as follows.

$$\mathbf{Y} = \begin{array}{|c|}\hline 1 \\ \hline 50 \\ \hline 70 \\ \hline 60 \\ \hline 70 \\ \hline 1 \\ \hline 0 \\ \hline \end{array}$$

Here Bx, By, Bw, and Bh values are given as an example.

Pc=1, probability of any class is 1 because there is one object

C1=1, because the object belongs to class ripe apple

C2 = 0, because there is no object from class green apple

When there is no object in the image Pc will be zero and the rest of the values do not matter.

When to train a neural network, classify the object as well as the bounding box of each of these images since it’s a supervised learning problem these need to give the bounding boxes. The neural network only understands numbers, they have to convert this into this kind of vector. A vector of size 7 for each corresponding image. The image is X-train and Y-train will be a vector of size 7. In general, there might be thousands of such images. A neural network can be trained in a way that when an input image is a feed to the network (figure 4.2), it will tell the particular vector and now this vector is telling that this is a ripe apple because C1=1 also gives the bounding box.

Multiple objects: The above method works only for a single object. For multiple objects, there could be an N number of objects in an object, it is hard to determine the dimension of the neural network output. If there 10 objects then have 10x7 vectors mean 70 size vector. That will not an efficient way. In Figure 4.6, there are two bounding boxes that the image has.

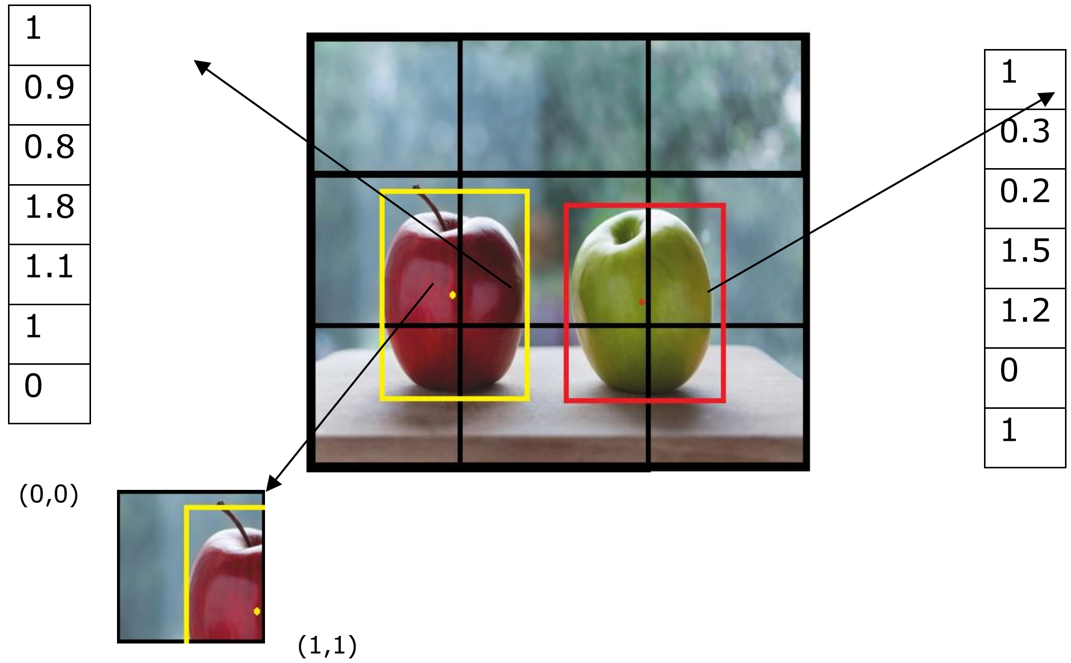


figure 4. 6 Encoding for multiple bounding boxes

In the center cell, there are two objects but the center point (red dot) of the green apple lies in a different cell. So, the green apple belongs to that cell.

The bounding rectangle that can go out of the grid cell, it is fine that's why Bh>1 and Bw>1.

Here the volume will be 3x3x7, For an image have 3x3 total grid cell 9 cells each cell is a vector of size 7.

The CNN will predict a y for each cell, so the size of the output tensor (multidimensional "matrix") should be:

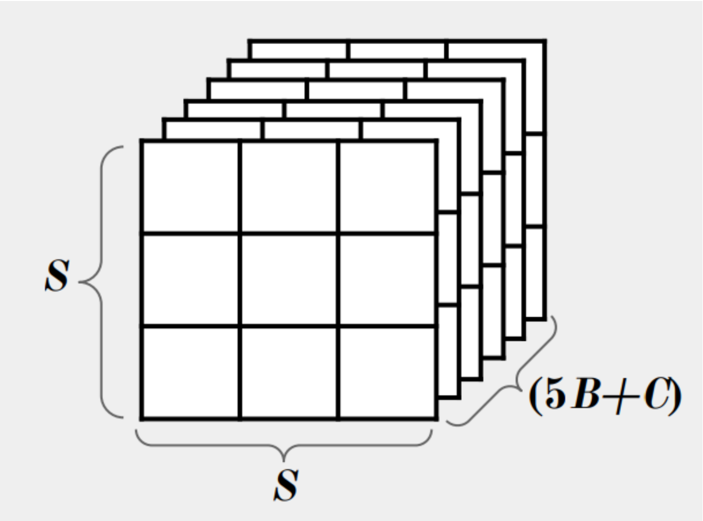


figure 4. 7 output feature map

size of the output tensor is calculated by

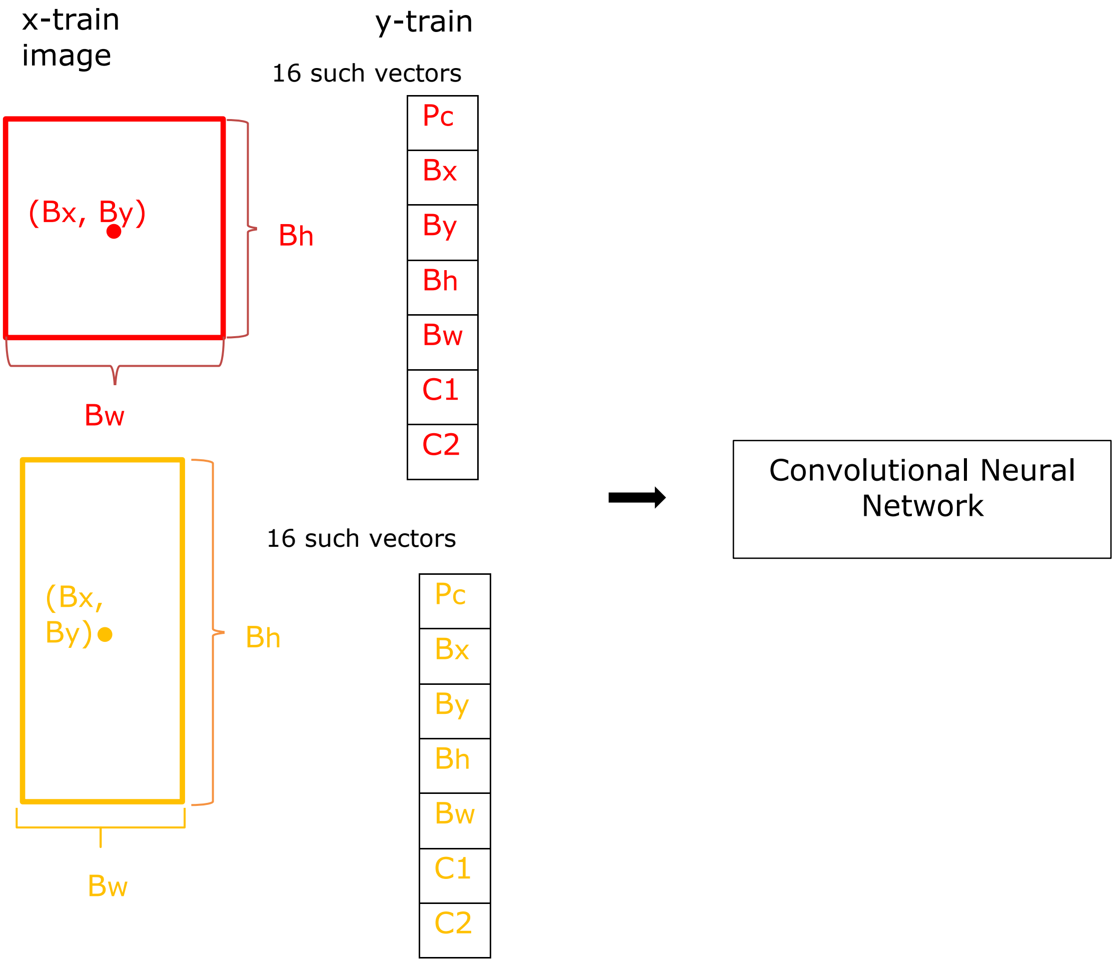
$$S \times S \times (5B+C) \tag{4.1}$$

where s=3, number of cells in an image

B=1, number of bounding boxes each cell can predict
C=2, number of classes

Size of output tensor = 3 x 3 x (1 x5+2)
=3 x 3 x 7 tensor

Now time to form the training dataset. The training data set will have so many such images. Each image has a bounding box and based on that bounding box ends up with a target vector y. for each cell there will be one vector. So, there will be 9 such vectors for training images. The next step is to train the neural network.



After trained it, it can do prediction. When trying to test the algorithm, It can produce 9 such vectors, which will show the bounding box for each of the objects. Therefore, it is called YOLO, you only look once because there is no repetition. It is not like repeat the process nine times. The output will be like the feature map illustrated in figure 4.8.

Multiple objects in a single grid cell: There could be another issue is what if a single cell contains the center of multiple objects, predict multiple bounding boxes per cell (B>1). This is often the case happens. This brings to the idea of anchor boxes. Consider the following image, which has been separated into a 3 x 3 grid:

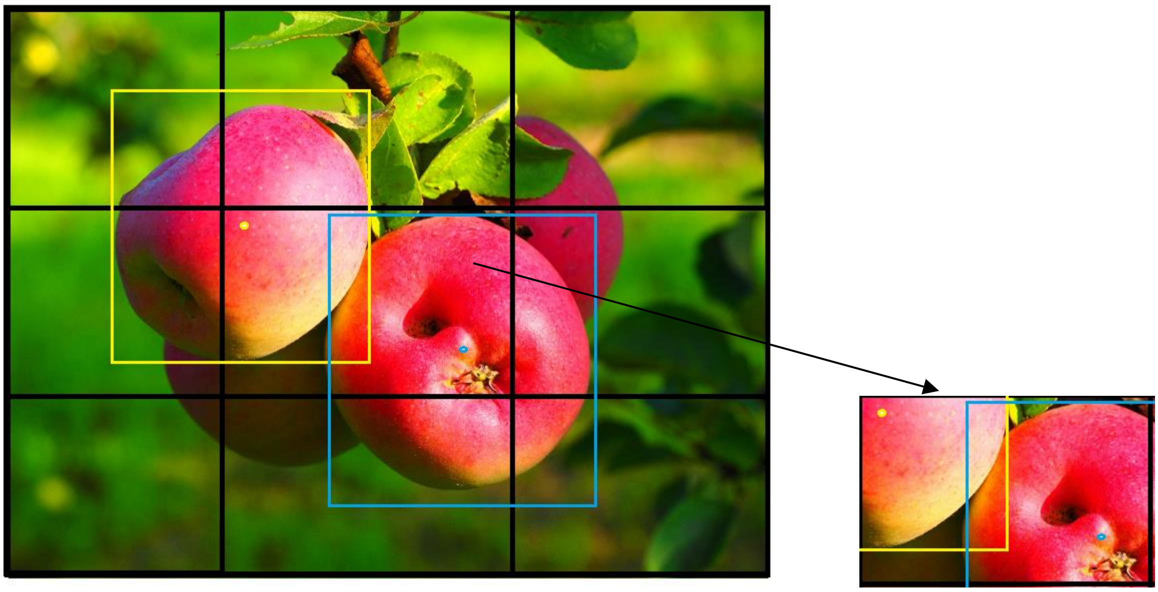
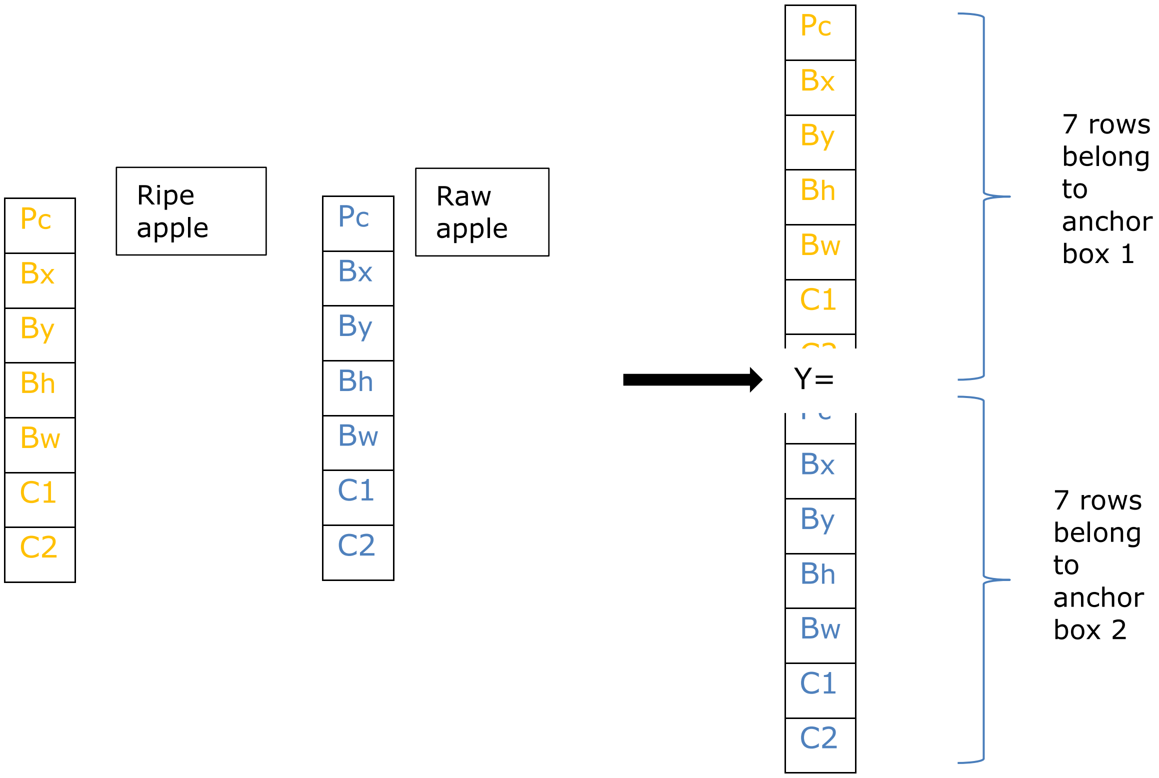


figure 4. 8 centers of two objects in one grid

Here in this case figure 4.9 the raw apple and the ripe apple both are in the middle grid cell. Both objects' midpoints are in the same grid in the example above. Now use the vector to represent the grid cell. but the vector can represent only one class.

Anchor box 1
Anchor box 2

The above boxes show the anchor boxes of both objects. to represent two class values for ripe apple and raw apple so, instead of having 7 dimension vector power here have a vector of size 14 as per the vector below, simply augment y.



This is said to have has 2 anchor boxes and this can have more than two anchor boxes. if there are three objects which have the same center there have been three anger boxes even have five boxes. but if the grid cells are small enough then in real life it's hard to have many objects belonging to one grid cell. Objects are allocated to anchor boxes depending on the shape of the bounding boxes and the shape of the anchor box. Instead of 3 X 3 X 7 (using a 3 X 3 grid and two classes), the production would be 3 X 3 X 14. (since using 2 anchors).

4.2.2 Intersection over Union and Non-Max Suppression:

There could be few issues with this approach first issue is the algorithm might detect multiple bounding rectangles for a given object it is possible. For one object the algorithm can detect multiple bounding boxes as shown in the picture below.

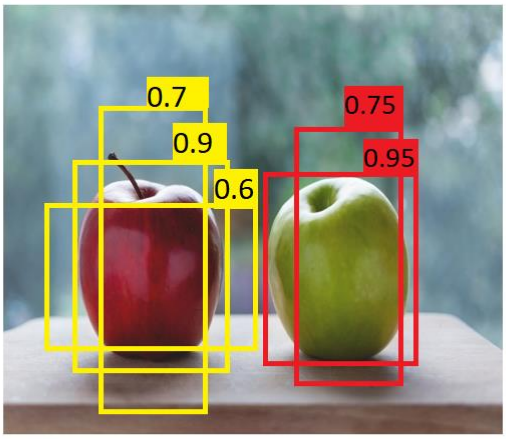


figure 4. 9 multiple bounding boxes

Consider figure 4.9, the red and green apples having multiple bounding boxes with a different confidence score. By visual observation, it is clear that the bounding box with the highest confidence score is the most accurate one. And the algorithm will also throughout the probability, Pc will predict the value. Other boxes have less probability. Maybe consider all the probabilities for a person's class and take the max, but this cannot be the right way. If we just take max and if there is another person, so neural networks don't know where the person is, it can not take max. This is where the concept of Intersection over Union comes into play. It computes the intersection of the actual bounding box and the predicted bounding box over their union.

Here, there are different bounding boxes with different probabilities, to decide it is a good prediction or not is a difficult task. So IoU, or Intersection over Union, will calculate the area of the intersection over a union of these two boxes. That area will be;

Upper left and lower right coordinates of a box: (*x1, y1, x2, y2*)

Coordinates of the intersection of two boxes: (*xi1, yi1, xi2, yi2*)

Where;

xi1 = maximum of the x1 coordinates of the two boxes
yi1 = maximum of the y1 coordinates of the two boxes
xi2 = minimum of the x2 coordinates of the two boxes
yi2 = minimum of the y2 coordinates of the two boxes

To calculate the area of a rectangle (or a box) we can multiply its height (y2 – y1) by its width (x2 – x1)

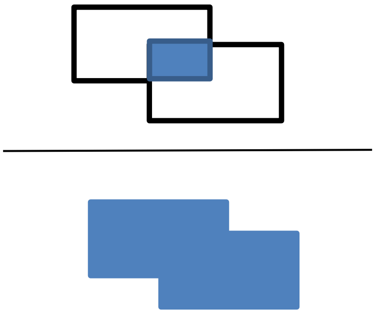
The area of intersection can be calculated by

$$(xi2 - xi1) \text{ X } (yi2 - yi1) \tag{4.2}$$

Area of union calculated by

$$(\text{area of box } 1 + \text{area of box } 2) - \text{Area of intersection} \tag{4.3}$$

$$\text{IoU} = \frac{Area\ of\ intersection}{Area\ of\ union}$$



Find division of these two and if the objects are overlapping intersection value will be more.

If it is the values more than 0.6 or 0.7, these rectangles are overlapping.

If there completely overlapping the value will be 1.

If they are not overlapping at all there will be 0.

In figure 4.9, find that these two yellow boxes are overlapping because their IoU greater than 0.65. so, then discard those rectangles. Discarded all the rectangles which had IoU greater than 0.65 and carry the rectangle which has class priority as Max. then do the same thing for green apple, find that 0.95 is the Max probability, and found all other rectangles in this image. Again, there will be more apples here and there will be rectangles for those also. So the neural network will try to find overlap. In case if there is another green apple here and the system will not find overlap it will not discard that rectangle. But this rectangle you find it to be overlapping and since 0.95 is Max .75 is less than discarding this and will get final bounding boxes. this technique is also called Non-max suppression so after the neural network has been detected all the objects then apply Non-max suppression and will get these unique bounding boxes

Intuitively, the higher the threshold, the more accurate the predictions become. Non-Max Suppression is another strategy that can greatly increase YOLO's performance.

Non-maximum Suppression (NMS) intends to cure the problem of multiple detections of the same image. One of the most common issues with object detection algorithms is that they may detect an object several times rather than only once. Here (fig 4.9,) The apples are recognized several times. This is cleaned up using the Non-Max Suppression technique, which results in just one detection per item. When the IoU between two boxes with the same label is above a certain threshold, non-max suppression solves multiple counting by eliminating the box with the lower confidence probability.

The following are the steps to implementing non-max suppression:

1. Identify the box with the highest confidence.
2. Calculate the overlap with all other boxes and eliminate all boxes that overlap it greater than a certain threshold (IoU threshold).
3. Repeat steps 1–3 until there are no more boxes with a lower score than the one selected.

Precision: it is the ratio of true positives (TP) to the total number of predicted positives (total predictions). The formula is given in this format.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4.4}$$

Where;

TP- True positives
FP – False positives

Recall: The ratio of true positives (correct predictions) to the overall number of ground truth positives is known as recall (total number of object). The formula is as follows:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4.5}$$

Where;

TP- True positives

FN – False negative

F1 Score: The F1-score is a metric for how accurate a model is on a given dataset**.** The F1 score represents a good balance of precision and recall**.**

$$\mathrm{F1} = 2 \text{ x } \frac{(\text{Precision x Recall})}{(\text{Precision} + \text{Recall})} \tag{4.6}$$

4.2.3 Average Precision and Mean Average Precision:

Calculating mAP (mean average precision) scores to test qualified models, which is a common metric for evaluating the accuracy of object detectors such as Faster R-CNN, SSD, Yolo, and so on. The average precision value is computed with recall values ranging from 0 to 1. The field under the precision-recall curve is a simple definition for the Average Precision. Precision and recall are combined in AP. It accepts a value between 0 and 1. (higher is better). Users need both precision and recall to be equal to 1 to get AP = 1. The mean of the AP estimated for all classes is the mAP. The mAP calculates a score by comparing the ground-truth bounding box to the observed box. The higher the score, the better the model's detection accuracy.

4.3 Object detection pipeline

The following object detection pipeline gives the detailed workflow of the object detection model to be developed. There are three major parts in developing the model.

1. Prepare image dataset

2. Build and train models

3. Model validation and testing

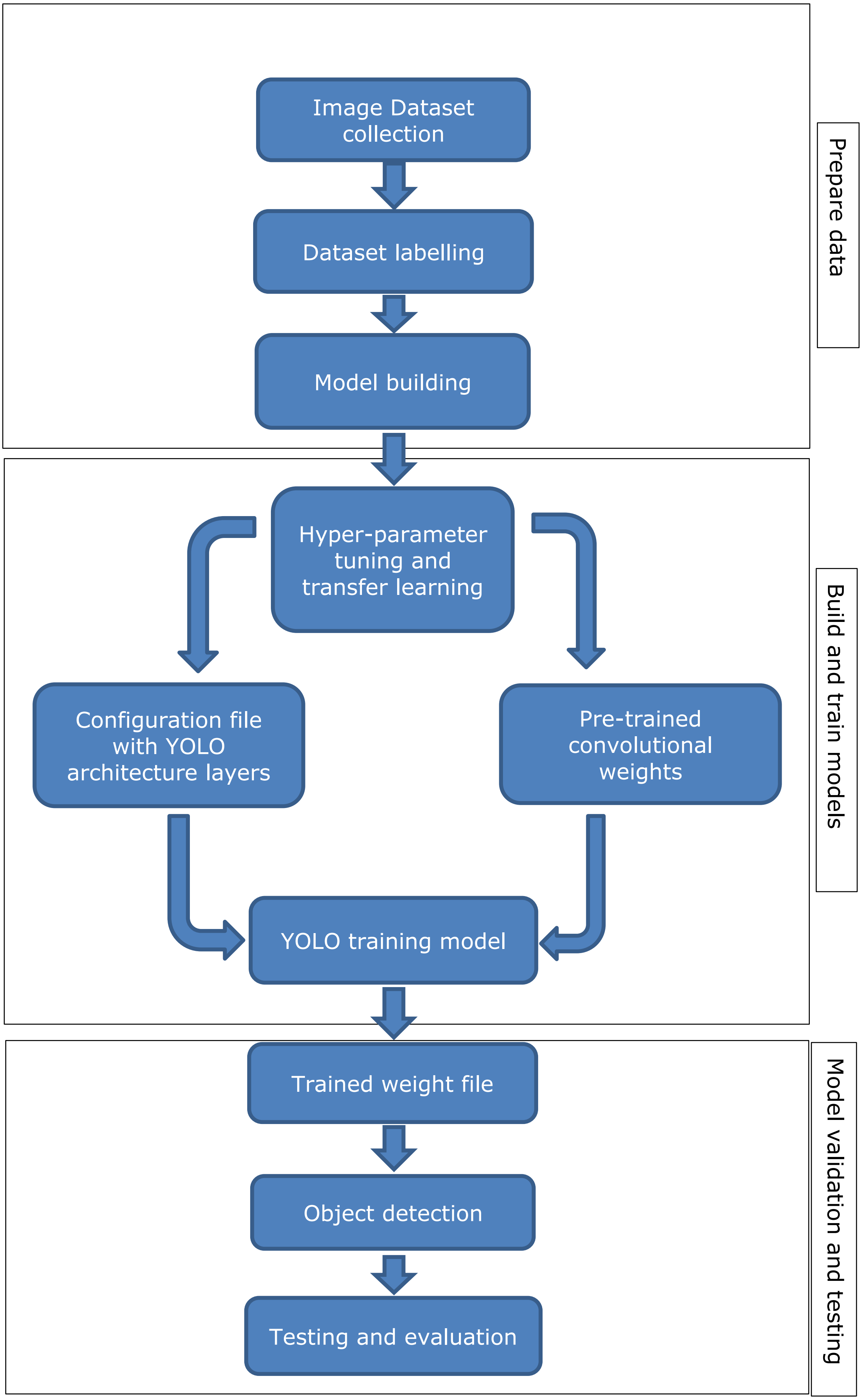


figure 4. 10 Object detection pipeline

4.4 Training YOLOv4 Apple Detector with Darknet in the Cloud GPU enabled

In this project, the author trains the Yolo "you only look once" version 4 object detector using Google Colab. The images of apples from different environments and lighting were used to train the model. Paula red and Gala apples are mainly used for testing the model. The detector trained to check whether the apple is ripe or not. Once the training is finished the detector can run over images, videos, and live streaming.

4.4.1 Dataset construction

The factors like illumination variation, occlusion, and overlap make the apple detection significantly difficult. To improve the accuracy dataset created by under different environments as follows:

- Single object with no occlusion
- Multiple objects with occlusion
- Clusters of apples
- Illumination variation
- Shading conditions
- Multiple objects with or without occlusion

4.4.2 Annotation of images with labelImg tool

To use Darknet to train YOLOv4 with the author's custom dataset, first, need to convert it to Darknet YOLO format. Here need to train a Custom object detection algorithm with help of a custom data set. So first step is to download images using an extension from Google. Then label all those images with the help of labelImg tool. For the project need to download 3 types of classes ripe apple, a raw apple, and a green apple. A good thing about these extensions is that it provides a filter for width and height of the images.

Annotations are the hardest part of this process. It is important to annotate all input images with four coordinates representing the location of the bounding box of an object and its label.

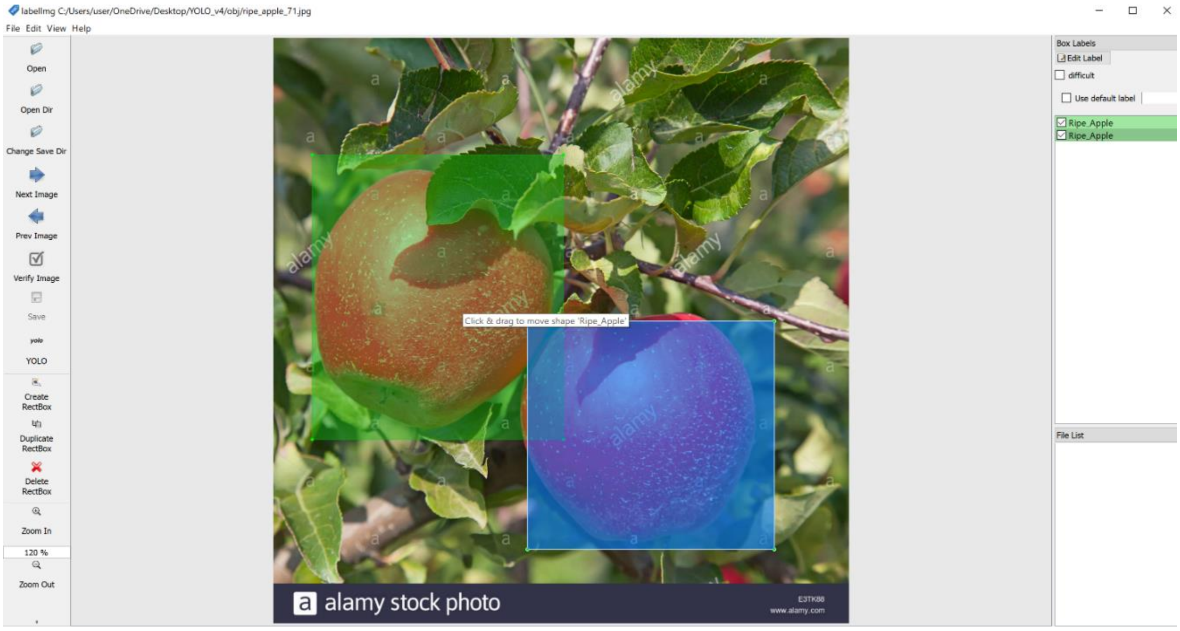
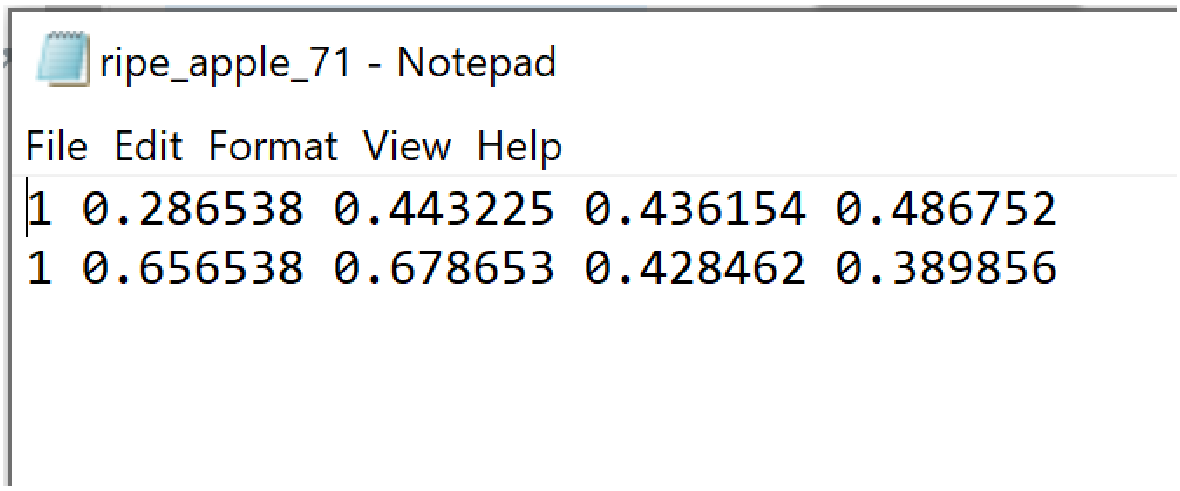


figure 4. 11 labelling images using labelImg

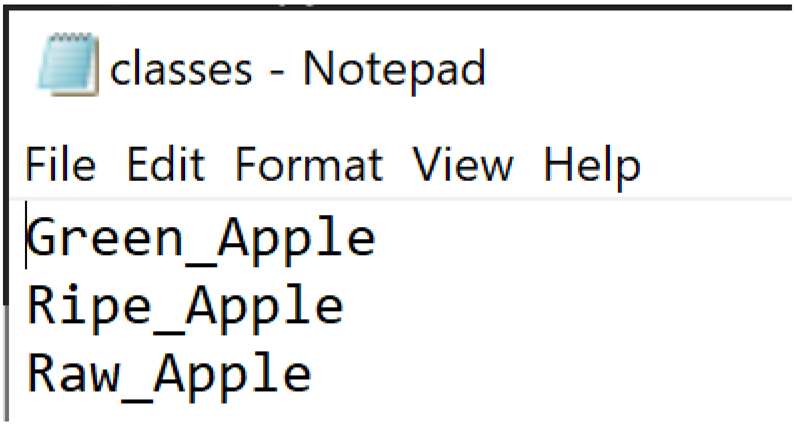
After completing annotation there should be one classes.txt and one .txt file in the same name of the image file in the directory shown in the image 4.12.



```
1 0.286538 0.443225 0.436154 0.486752
1 0.656538 0.678653 0.428462 0.389856
```

figure 4. 12 YOLO label file

There are five values in the YOLO label file. The first value is the class index id, as defined in the classest.txt file. The index begins at zero (zero). The 0 value indicates that this is the very first class. Figure 4.12 shows that it is 1 since it is the second class, “Ripe Apple.” The object coordinates, as well as the height and width of the bounding box, are the remaining four values in the register.



classes - Notepad

File Edit Format View Help

```
Green_Apple
Ripe_Apple
Raw_Apple
```

figure 4. 13 classes file

Figure 4.13 shows the classes.txt file that contains the number of classes used in the project.

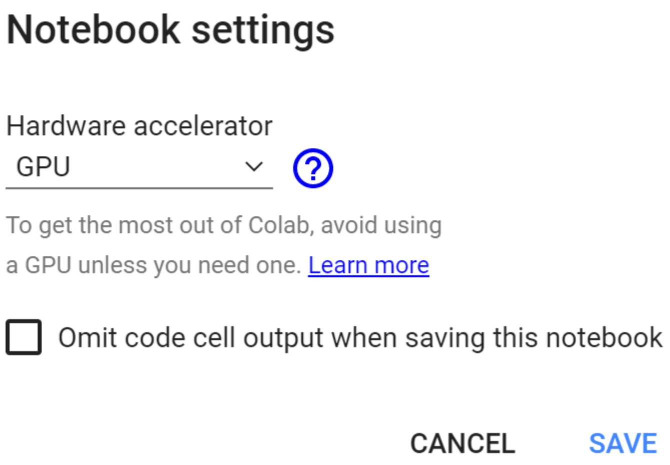
4.4.3 Setting up Google Colab as a cloud VM with free GPU

For computing, the author going to use Google Colab. The process required to enable GPU acceleration within our Colab notebook so that our YOLOv4 system will be able to process detections over 100 times faster than CPU. the important things that will need to train YOLOv4 are the following:

- GPU with specific GPU drivers installed
- OpenCV
- cuDNN configured on top of GPU drivers

It's simple to prototype and run the benchmark framework since Google Colab comes preconfigured with GPU, OpenCV, and CUDNN Cuda libraries.

Once Completed the labeling of the images now is the time to upload this complete data set into Google Drive. The author will be utilizing Google Drive as external storage for the Google colab. So, create a zip file based on the images have labeled then create a new folder named YOLOv4 model training inside Google Drive. Next, create another folder named 'training' inside the YOLOv4 folder. This is where the trained weights will save (this path is mentioned in the obj.data file which will be upload later). Go to google colab and start a new notebook in the name of YOLOv4_Apple_Detector. Then set up the google colab author need to enable the GPU. Once connected the colab then changes run-time type to GPU.



Notebook settings

Hardware accelerator

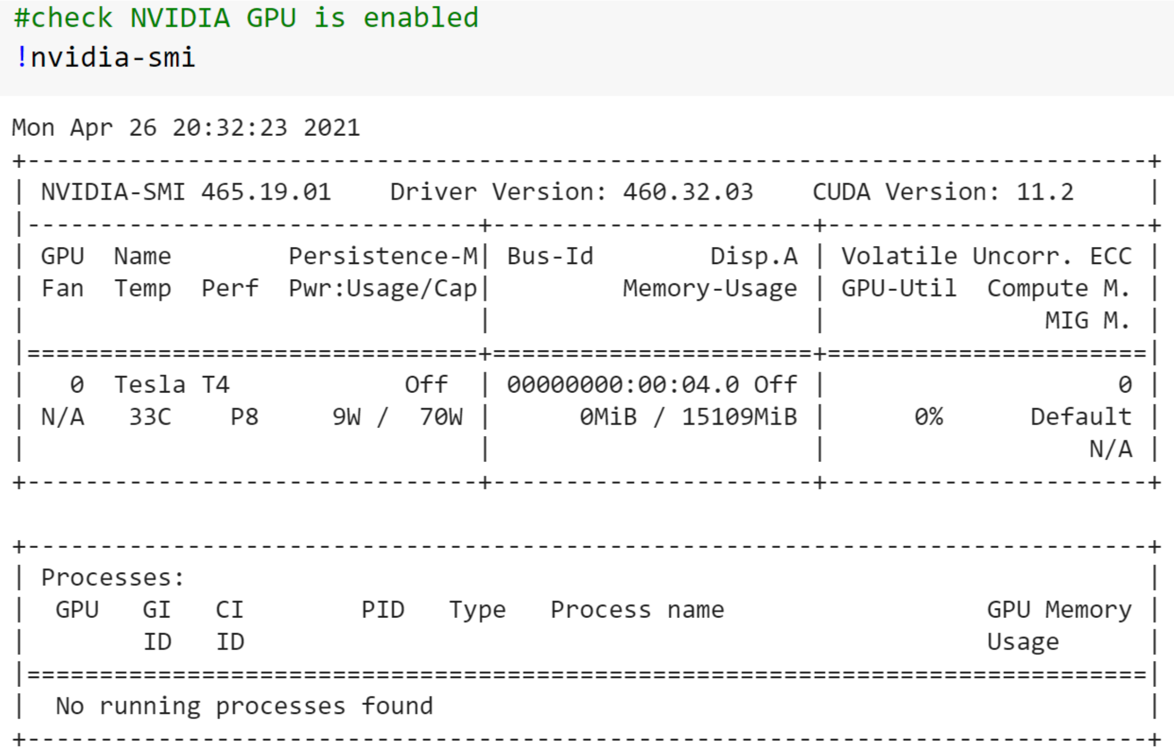
GPU

To get the most out of Colab, avoid using a GPU unless you need one. Learn more

□ Omit code cell output when saving this notebook

figure 4. 14 Notebook settings

To check if NVIDIA GPU is enabled as follows;



```
#check NVIDIA GPU is enabled
!nvidia-smi
```

```
Mon Apr 26 20:32:23 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 465.19.01    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   33C    P8     9W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

figure 4. 15 Received GPU

From figure 4.15, this time from colab received the NVIDIA Tesla T4 GPU which is quite fast for our training. Next is to munt google drive to google colab and navigate to /mydrive/yolov4 folder.

```
# mount drive
%cd ..
from google.colab import drive
drive.mount('/content/gdrive')

# this creates a symbolic link so that now the path /content/gdrive/My\ Drive/ is equal to /mydrive
!ln -s /content/gdrive/My\ Drive/ /mydrive

# list the contents of /mydrive
!ls /mydrive

#Navigate to /mydrive/yolov4
%cd /mydrive/yolov4
```

Here the author used the '!ls' shell command to check how many directories are there in the folder and the %cd command to move within the directory.

4.4.4 Configuring files for training in the cloud

In this project, object detection utilizes darknet in the background. Once the notebook and drive settings are finished then need to clone the darknet GitHub repo into the yolov4 folder in the drive.

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

This is going to clone the complete repository on the router address to the specified folder inside the drive. The next step is to create and upload the following files which need for training the apple detector.

a) Labeled Custom Dataset

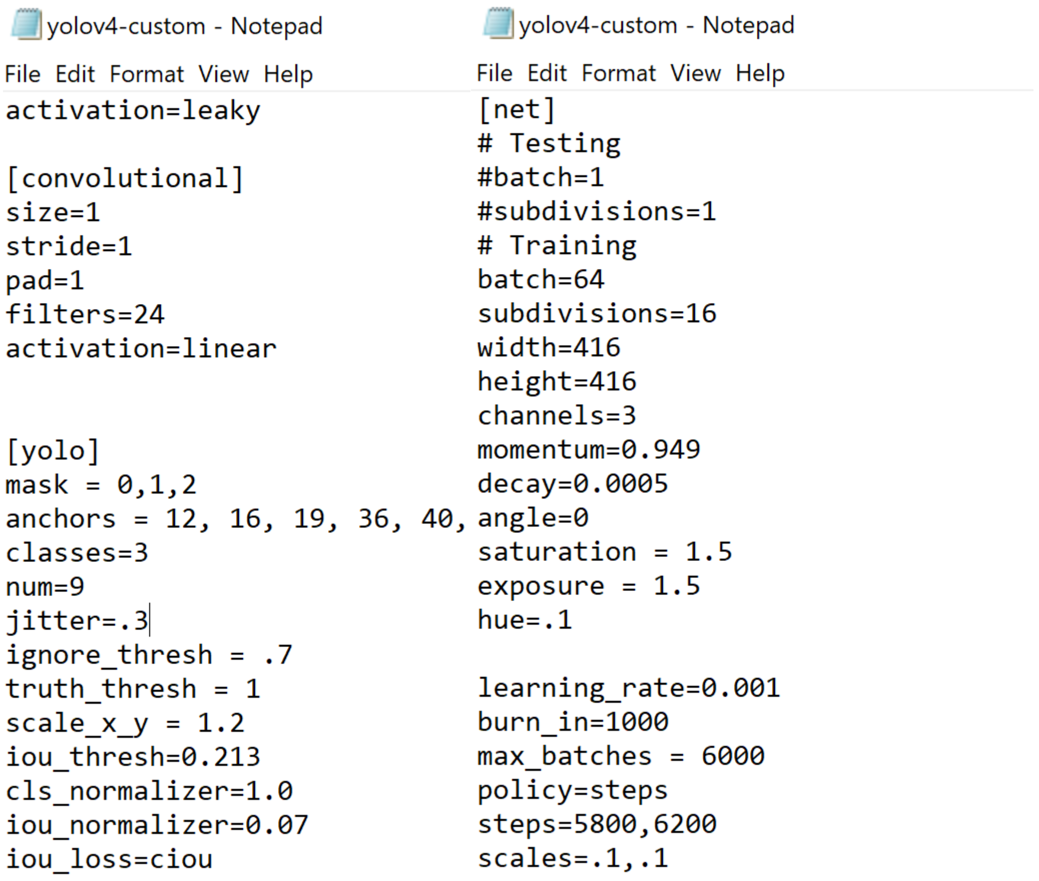
b) Custom cfg file

c) obj.data and obj.names files

d) process.py file (to make the files train.txt and test.txt for training)

a) **labeled custom dataset**: Create the 'obj.zip' zip file, which contains both the input image ".jpg" files and their corresponding YOLO format labeled ".txt" files, from the obj folder. Upload the zip file to Google Drive's yolov4 folder.

b) **Custom cfg file:** Configuring the training for YOLOv4 for a custom dataset is a tricky process. First of all download the "yolov4-custom.cfg" file from darknet/cfg directory, make the following changes, and upload it to the yolov4 folder inside the drive.



```
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=24
activation=linear


[yolo]
mask = 0,1,2
anchors = 12, 16, 19, 36, 40,
classes=3
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
scale_x_y = 1.2
iou_thresh=0.213
cls_normalizer=1.0
iou_normalizer=0.07
iou_loss=ciou
```

```
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=16
width=416
height=416
channels=3
momentum=0.949
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 6000
policy=steps
steps=5800,6200
scales=.1,.1
```

figure 4. 16 custom cfg file

- change line batch to **batch=64,** the batch size is the number of images per iteration
- change line subdivisions to **subdivisions=16,** subdivisions are the number of pieces of a batch is broken into for GPU memory.
- change line **max_batches to 6000** (classes*2000 but not less than number of training images and not less than 6000)

- Set line steps to 80% and 90% of max batches respectively, i.e **steps = 5800, 6200**
- Change the network size to **width=416 and height=416** (or any multiple of 32). These can be any multiple of 32, with 416 being the most common. Increasing the value to a higher number, such as 608, will sometimes increase performance, but it can slow down the training.
- change line classes=80 to **classes = 3** number of objects in each of 3 [yolo]-layers
- change [filters=255] to **filters = 24**

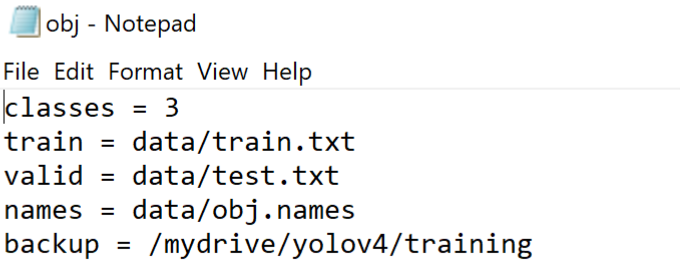
$$\text{filters}= (\text{classes} + 5)\ \text{x}3 \qquad (4.6)$$

$$\text{filters}= (3+5)\ \text{x}\ 3 = 24\ \text{filters}$$

In the 3 [convolutional] before each [yolo] layer, keep in mind that it only has to be the last [convolutional] before each of the [yolo] layers. In this case, changed the classes 3 in the three YOLO layers and filters 24 in the three convolutional layers before the YOLO layers.

c) obj.data and obj.names files:

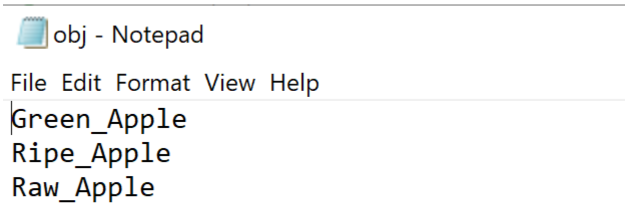
Create an obj. data file and fill it in like the figure 4.17. it contains the number of classes, a path to the test and train files, a path to object name file which has all the class names in it also contain the backup path where all the weights of the model are saved throughout the training process. Create a backup folder in google drive and put its correct path in this file.



```
classes = 3
train = data/train.txt
valid = data/test.txt
names = data/obj.names
backup = /mydrive/yolov4/training
```

figure 4. 17 obj. data file

Also, need to create a new file within a code or text editor called obj. names that have one class name per line in the same order as in the classes.txt from the dataset generation step.



```
Green_Apple
Ripe_Apple
Raw_Apple
```

figure 4. 18 obj. names file

Then upload both obj.data and obj.names file to the drive

d) Process.py file:

The py script generates the files train.txt and test.txt, with train.txt containing paths to 90% of the images and test.txt containing paths to 10% of the images.

4.4.5 Build darknet

In this step make some changes in the make file in the darknet folder. There are multiple options like GPU, CUDNN, OpenCV, and so on. This project requires utilizing some of the parameters. So in the properties, we have to make the following changes.

```
# change makefile to have GPU and OPENCV enabled
# also set CUDNN, CUDNN_HALF and LIBSO to 1

%cd darknet/
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
!sed -i 's/LIBSO=0/LIBSO=1/' Makefile

# set GPU=1 and CUDNN=1 to speedup on GPU
# set CUDNN_HALF=1 to further speedup 3 x times (Mixed
precision on Tensor Cores) GPU: Volta, Xavier, Turing and higher
```

Enable CUDNN, CUDNN_HALF, and LIBSO. LIBSO going to allow us to get access to darknet using Python so that's key. Colab occasionally shifts dependencies around, at the time of authorship this makefile works for building darknet on colab. The further step is to run the make command this is going to build all the binaries files and get darknet properly running so that can access the darknet.py file and run its Python functions to have the yolov4 detection's run.

```
# build darknet
!make
```

The building process will not be interrupted once it gets interrupted there should be a chance to change the directory.

Following that, copy all of the files from the yolov4 folder to the darknet directory. /mydrive/yolov4/darknet is the new working directory.

Except for the labels folder within the data folder, which is necessary for writing label names on the detection boxes, clean the data and cfg files. The directory contains an excessive number of unnecessary files. It will be easier to navigate if it is cleaned.

So just delete all other files from the data folder and clean up the cfg folder; the custom config file is already in the yolov4 folder on Google Drive.

```
# Clean the data and cfg folders first except the labels folder in data which is required

%cd data/
!find -maxdepth 1 -type f -exec rm -rf {} \;
%cd ..

%rm -rf cfg/
%mkdir cfg
```

There are several steps to copy every file as follows;

a) Unzip the obj.zip dataset and its contents so that they are now in /darknet/data/ folder.

```
!unzip /mydrive/yolov4/obj.zip -d data/
```

b) Copy the yolov4-custom.cfg file so that it is now in /daknet/cfg/ folder.

```
!cp /mydrive/yolov4/yolov4-custom.cfg cfg

# verify if your custom file is in cfg folder
!ls cfg/
```

After copying, verified the file location by !ls command.

c) Copy the obj.names and obj.data files from the google drive so that they are now in /darknet/data folder. Then check the directory using shell command.

d) The last step is to copy the process.py file to the current darknet directory.

4.5 Training custom detector

4.5.1 Generating train.txt and test.txt files

In order to create the train.txt and test.txt files, it requires to run the command to run process.py. This will create train.txt and test.txt files inside the darknet/data folder. Also, it is good way to check the directory whether the file is inside the folder or not by using a shell command. List the contents of the data folder to check if the train.txt & test.txt files have been created.

```
# run process.py (this creates the train.txt and test.txt files in our darknet/data
 folder )
!python process.py
```

The above process.py creates the two files train.txt and test.txt. where train.txt has path to 90% of the images and test.txt has 10% of the images. In this case two hundred images for train.txt and 22 images for test.txt.

4.5.2 download the pre-trained yolov4 weights

For next step is to download the pre-trained weights. Here author need to take an approach called transfer learning. Download a model which is pre trained for some sort of classes and the author try to optimize on top of that model with the custom image data set used in this project. For this project using pre-trained YOLOv4 weights which have been trained up to 137 convolutional layers. Run the following command to download the YOLOv4 pre-trained weights file from the repository.

```
# Download the yolov4 pre-trained weights file
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_
optimal/yolov4.conv.137
```

The neural network has three sorts of yolo layers. In here supposed to change the last few of layers of the transfer learn model to configure it for the custom dataset. That's why in the configuration file changed the last 3 yolo layers and preceding conv layers (includes classes and filters) to this three yolo layers. YOLOv4 has been trained already on the coco dataset which has 80 classes that it can predict. Grab these pretrained weights so that can run YOLOv4 on these pretrained classes and get detections. So, this model which is pre trained for some sort of classes now change this model for this project's purposes. Utilize this model and then transfer learning model to train on top of the classes that are created in this project. This step downloads the weights for the convolutional layers of the YOLOv4 network. By using these weights, it helps author's custom object detector to be way more accurate and not have to train as long.

4.5.3 Training the model

Now it is ready to start training the custom model yolov4 apple detector over the classes that specified. This training could take several hours depending on iterations chose in the configuration file. As per the custom configuration file it has 6000 iterations. train the model until the average loss does not show any significant change for a while.

```
# train our custom detector!
!./darknet detector train data/obj.data cfg/yolov4-custom.cfg yolov4.conv.137 -
dont_show -map
```

Considering the above command giving path to the detector detector and then the train, if it is for testing the model then have to write test over here. Then giving the relative path to the obj.data file. Then call the edited cfg file, also putting down the pre-trained weights. In the command '-dont_show' flag stops chart from popping up since Colab Notebook can't open images on the spot, '-map flag' overlays mean average precision on chart to see the accuracy of the model,the map flag added for the validation dataset. Sometimes, the runtime just dies intermittently. the only disadvantage, colab GPU can use it for 12 hours in a row, after that it will be disconnected. As it takes 6000 iterations to finish the training. The mAP parameter gives the mean average precision.. There are several interruptions during training. So saving the trained weights are more important. After each 100 iterations, the weights are stored as yolov4-custom last. Weights in the yolov4/training folder inside the google drive (The backup route in the "obj.data" file.). From the last trained weights can resume the training from where the process left.

4.5.4 Check performance from training chart

The performance can be verified either by chart or by checking the mean average precision.. So the first method is to verify the performance by analyzing the graph.

Before that specify the helper function 'imShow'. For that need to import some of the basic libraries cv2, matplotlib. Read the image using cv2 by giving path. Matplotlib to show the image within the notebook. Instead of using OpenCV to plot images, choosing Matplotlib because it's a little more difficult to visualize in a notebook. Matplotlib utilizes RGB channels while cv2 utilizes BGR format to read the image. Next to define the height and width of the image and RGB channel (which is 3). So the image has width, height and channel but yolo net takes the image in different format. So it required to convert the image in different format by the cubic interpolation method. The helper function can provide the resulting color image with a desired shape.

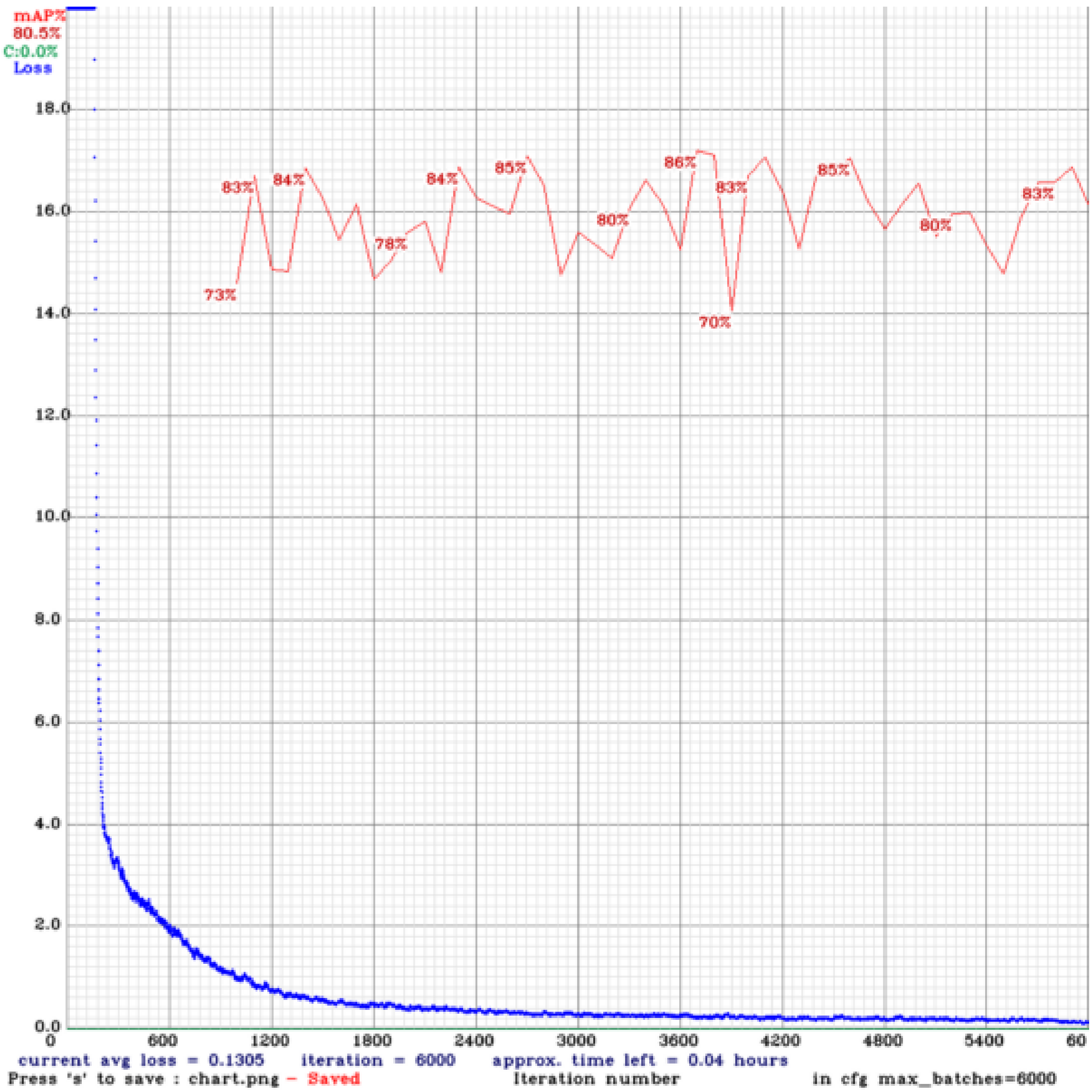


figure 4. 19 performance graph (average loss vs iterations)

Figure 4.19 shows a chart of average loss vs. iterations. For the custom detector model to be accurate, it should aim for a loss under 2. During the training the loss value went up quickly going down and after a number of iterations its going to steady off. And every 1000 iterations it plot mAP value it went up quickly and then it starts to be recede and it matches up with the loss and the accuracy increase that much loss is barely going down. This is because of overfitting. The training should not be stopped unless the certain to overfit. The spot where the early stopping point where will not be getting better model for the training.

4.5.5 Check the mAP (Mean average precision)

The performance can be verified by checking the mean average precision of each saved weight file that obtained from the training. The weights files are saved after each thousand iterations. In total there are 6000 iterations. Also have the best weights file inside backup folder. The higher the mAP the better it is for object detection. If the final weight file has overfitted then it is important to run these mAP commands to see if one of the previously saved weights is a more accurate model for the custom defined classes. Run the following command to check the mAP for each and every saved weights.

```
##WE can check the mAP for all the saved weights to see which gives the best res-
ults ( yolov4-custom_best.weights here is the saved weight change
the number like 4000, 5000 or 6000 snd so on )

!./darknet detector map data/obj.data cfg/yolov4-
custom.cfg /mydrive/yolov4/training/yolov4-custom_1000.weights -points 0
```

The results for each weight are given below in table 4.1.

Custom weights	**Avg IoU(%)**	**mAP (%)**	**Green apple, Ap (%)**	**Ripe apple, Ap (%)**	**Raw apple, Ap (%)**
1000	64.19	88.89	75.45	93.22	98
2000	78.76	99.93	100	100	99.8
3000	79.36	96.62	90.07	100	99.8
4000	85.34	96.43	92.68	100	100
5000	83.96	100	100	100	100
6000	89.10	100	100	100	100
Best weights	63.24	85.81	84.83	86.98	85.62

Table 4. 1 performance results

Custom weights	**Threshold value**	**Precision (%)**	**Recall (%)**	**F1 score(%)**
1000	0.25	85	93	89
2000	0.25	98	100	99
3000	0.25	95	100	98
4000	0.25	98	100	99
5000	0.25	97	100	98
6000	0.25	98	100	99
Best weights	0.25	78	81	80

Table 4. 2 Model performance evaluation with a 0.25 dataset at a resolution of 416 x 416 pixels

Since around 4000 area the getting overfit, the average loss was not going down.

4.6 Testing custom detector

Once the training finished, next step is to test the model. Before committing testing there should have to make some changes in the custom configuration file. For training the batch and subdivisions are changed to 64 and 16 respectively. In testing both parameters should be 1. This can be done either manually or by a command.

- change line batch to batch=1
- change line subdivisions to subdivisions=1

4.6.1 Run apple detector on image

For testing the model build and install OpenCV from source code using visual studio and cmake. By installing CMake can configure generate binary source code. Then installed visual studio to build source code. Building and installing OpenCV from source and verifying it works.

Next step is to darknet build for YOLO version 4. Downloaded the darknet repository to the PC. Form program files NVIDIA GPU computing copy cuDNN.64_7.dll and openv build.dll files to the darknet build folder. Then modify 2 files (Yolo_CP_DLL .VCX, Yolo_CPP_DLL.VCX) inside the darknet target folder change it to the version of CUDA. Here the author using CUDA 10.1. once it finished open the build file using visual studio and change properties enable OpenCV path to the visual studio and change the directories as per the requirement in the properties, Then build darknet.

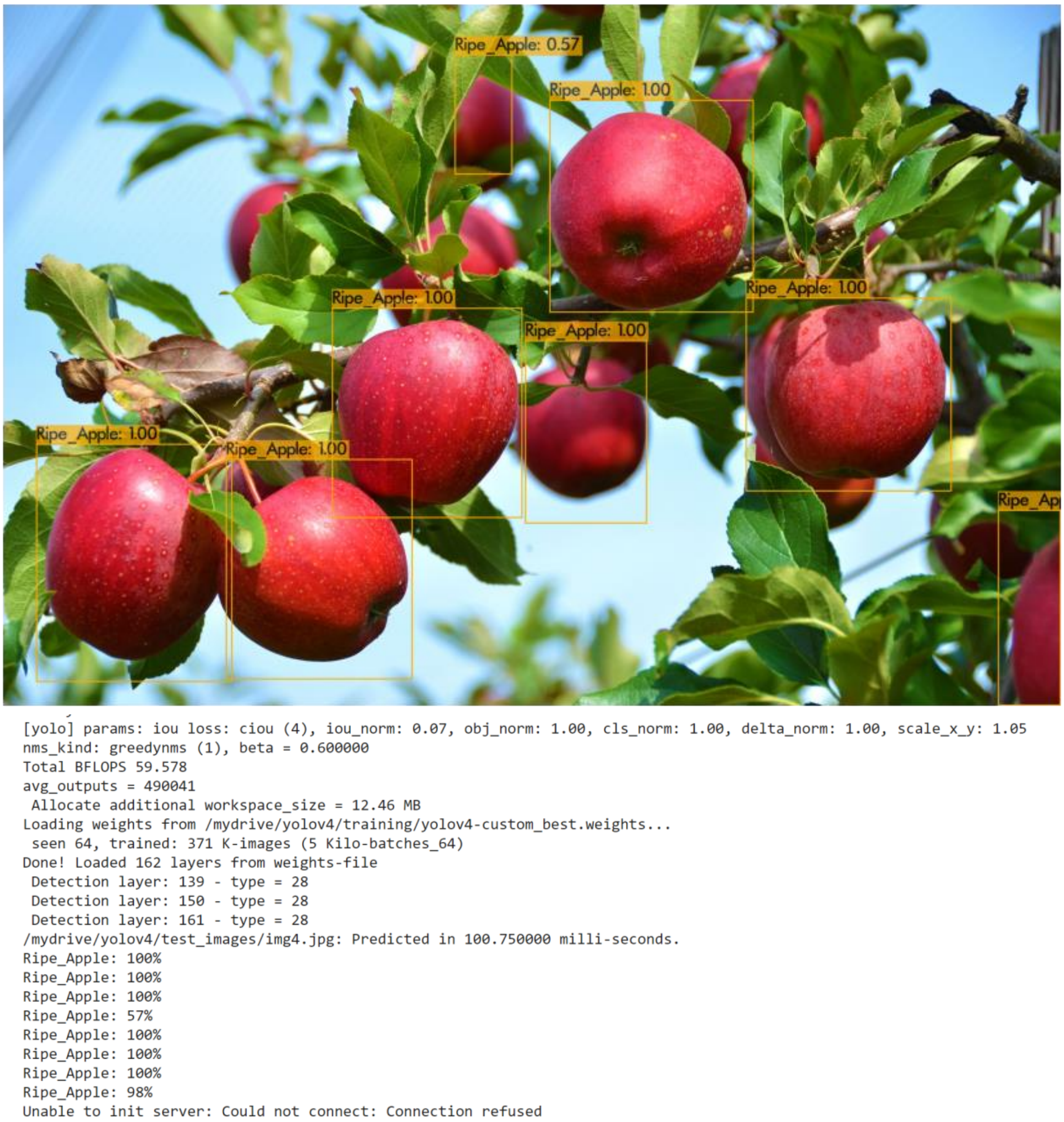
To run object detection on yolov4 need to download the weight files that trained before. The weight files copied to the darknet folder. Everything set for detection.

For testing images from open image dataset and free stock photos. By using the following command tried to run the custom detector on image. The thresh flag sets the minimum accuracy required for object detection. In here for testing the custom detector sets the threshold value to 0.3.

Then run the command below;

```
# run your custom detector with this command
!./darknet detector test data/obj.data cfg/yolov4custom.cfg /mydrive/yolov4/traini
ng/yolov4-custom_best.weights /mydrive/yolov4/test_images/img4.jpg -
thresh 0.3
imShow('predictions.jpg')
```

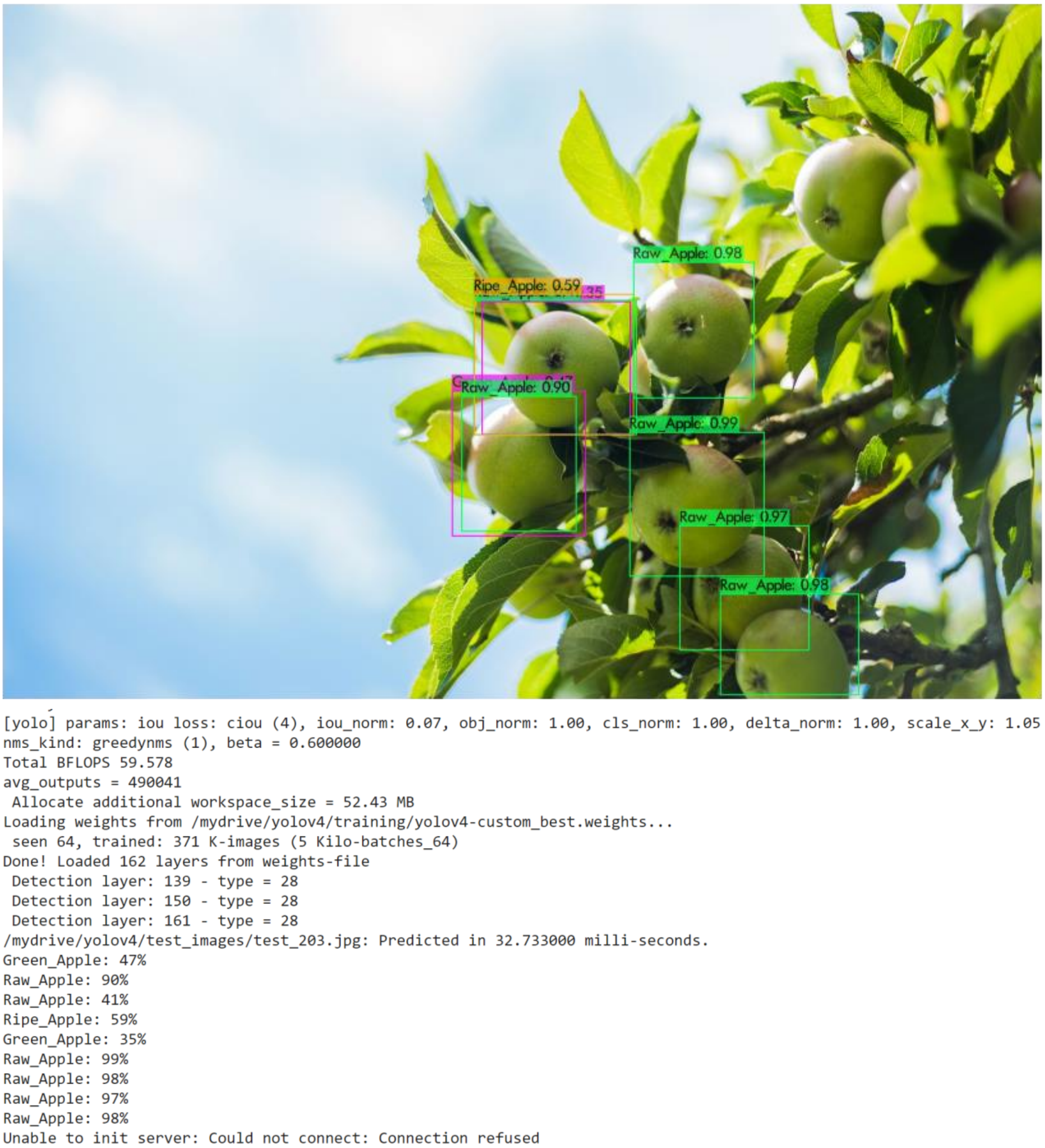
The following are the different test results:



```
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05
nms_kind: greedynms (1), beta = 0.600000
Total BFLOPS 59.578
avg_outputs = 490041
 Allocate additional workspace_size = 12.46 MB
Loading weights from /mydrive/yolov4/training/yolov4-custom_best.weights...
 seen 64, trained: 371 K-images (5 Kilo-batches_64)
Done! Loaded 162 layers from weights-file
 Detection layer: 139 - type = 28
 Detection layer: 150 - type = 28
 Detection layer: 161 - type = 28
/mydrive/yolov4/test_images/img4.jpg: Predicted in 100.750000 milli-seconds.
Ripe_Apple: 100%
Ripe_Apple: 100%
Ripe_Apple: 100%
Ripe_Apple: 57%
Ripe_Apple: 100%
Ripe_Apple: 100%
Ripe_Apple: 100%
Ripe_Apple: 98%
Unable to init server: Could not connect: Connection refused
```

figure 4. 20 Test result 1:multiple objects with or without occlusion

From the figure its shows the prediction completed within 100.75 milli-seconds. the image contains multiple object with or without occlusion.

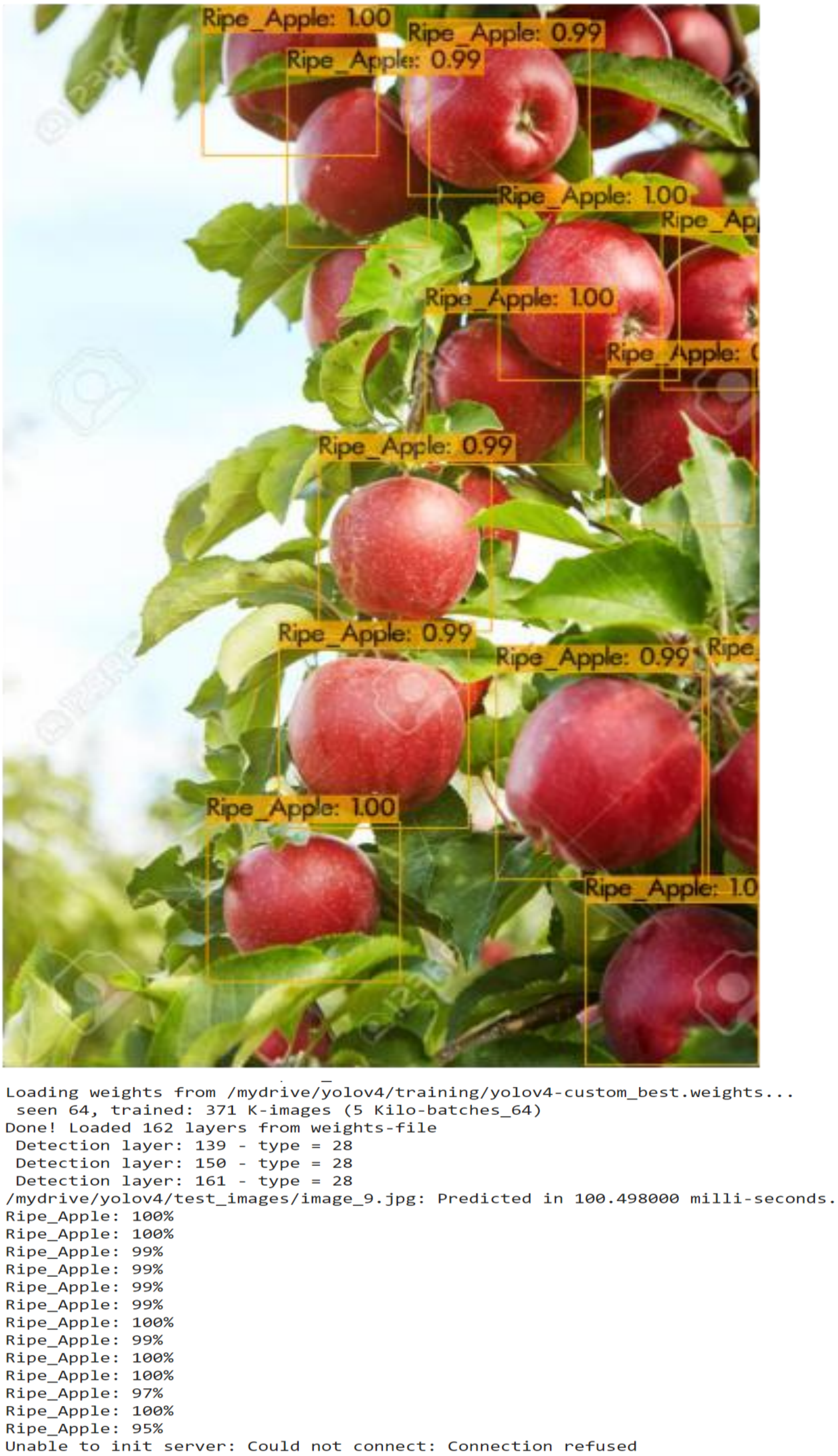


```
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05
nms_kind: greedynms (1), beta = 0.600000
Total BFLOPS 59.578
avg_outputs = 490041
 Allocate additional workspace_size = 52.43 MB
Loading weights from /mydrive/yolov4/training/yolov4-custom_best.weights...
 seen 64, trained: 371 K-images (5 Kilo-batches_64)
Done! Loaded 162 layers from weights-file
 Detection layer: 139 - type = 28
 Detection layer: 150 - type = 28
 Detection layer: 161 - type = 28
/mydrive/yolov4/test_images/test_203.jpg: Predicted in 32.733000 milli-seconds.
Green_Apple: 47%
Raw_Apple: 90%
Raw_Apple: 41%
Ripe_Apple: 59%
Green_Apple: 35%
Raw_Apple: 99%
Raw_Apple: 98%
Raw_Apple: 97%
Raw_Apple: 98%
Unable to init server: Could not connect: Connection refused
```

figure 4. 21 Test result 2: illumination variation

Figure 4.21 consist of an image with high illumination and a shading condition. The prediction completed within 32.73 milliseconds. Each and every classes were detected successfully with the confidence score.

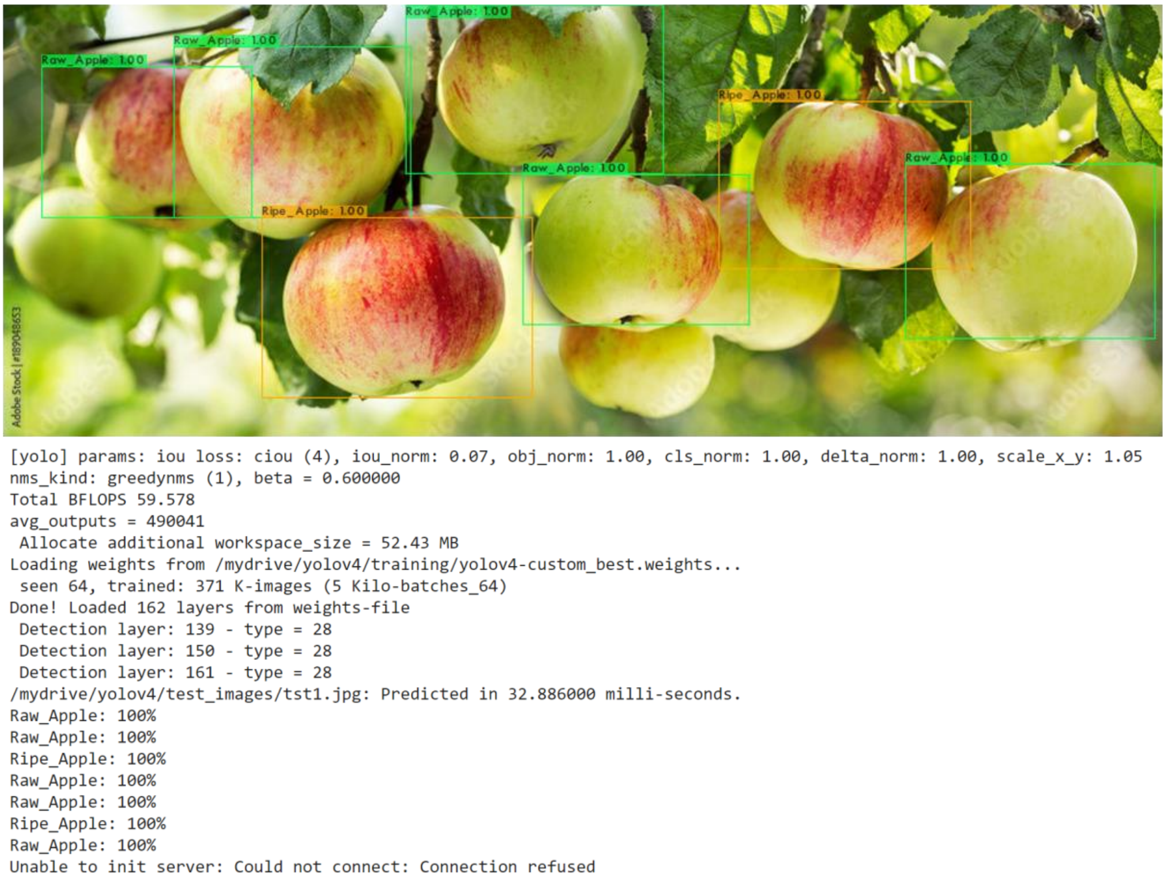
One of the major issues that is faced in apple detection is occultation. So the test image 3 is fed to the model for testing. Then the result shows in figure 4.22, most of the apples were detected with high accuracy even some of them are covered by leaves even though they were detected accurately.



```
Loading weights from /mydrive/yolov4/training/yolov4-custom_best.weights...
 seen 64, trained: 371 K-images (5 Kilo-batches_64)
Done! Loaded 162 layers from weights-file
 Detection layer: 139 - type = 28
 Detection layer: 150 - type = 28
 Detection layer: 161 - type = 28
/mydrive/yolov4/test_images/image_9.jpg: Predicted in 100.498000 milli-seconds.
Ripe_Apple: 100%
Ripe_Apple: 100%
Ripe_Apple: 99%
Ripe_Apple: 99%
Ripe_Apple: 99%
Ripe_Apple: 99%
Ripe_Apple: 100%
Ripe_Apple: 99%
Ripe_Apple: 100%
Ripe_Apple: 100%
Ripe_Apple: 97%
Ripe_Apple: 100%
Ripe_Apple: 95%
Unable to init server: Could not connect: Connection refused
```

figure 4. 22 Test result 3: clusters of apples

From test result 3 figure 4.21 has given the idea that YOLO apple detector can identify each apples individually from a group of apples. That would be very beneficial in real time detection. From the results it is clear that the yolo apple detector can run detection over a collection of apples in a apple tree more precisely.



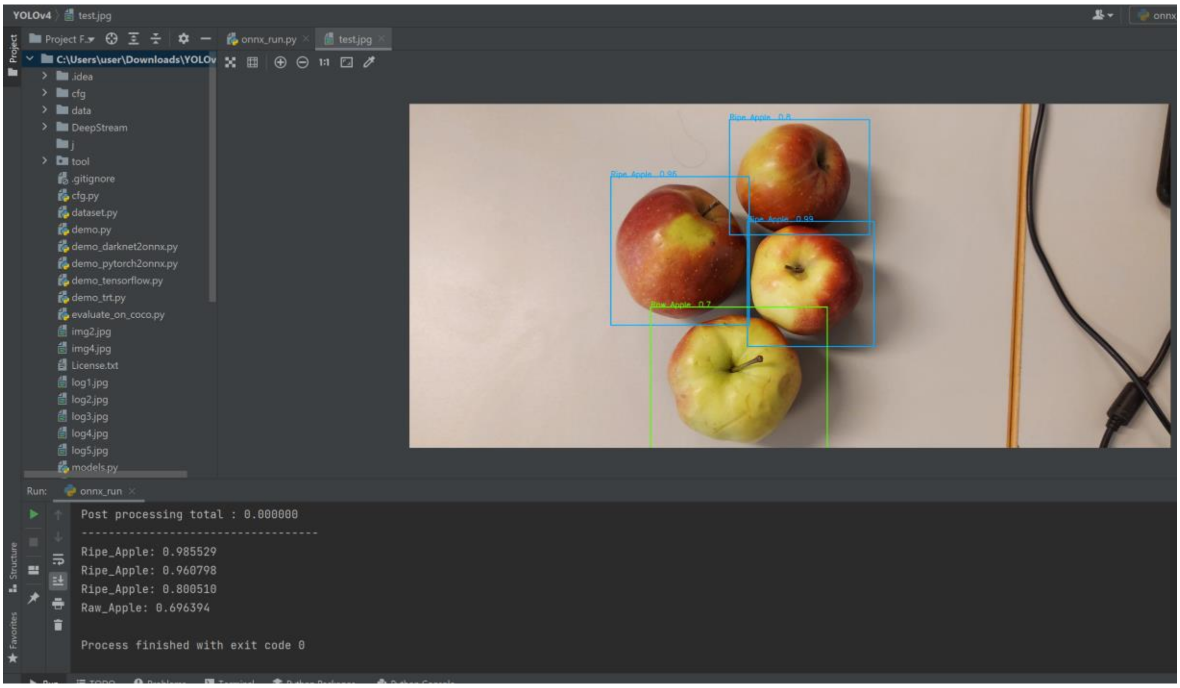
```
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05
nms_kind: greedynms (1), beta = 0.600000
Total BFLOPS 59.578
avg_outputs = 490041
 Allocate additional workspace_size = 52.43 MB
Loading weights from /mydrive/yolov4/training/yolov4-custom_best.weights...
 seen 64, trained: 371 K-images (5 Kilo-batches_64)
Done! Loaded 162 layers from weights-file
 Detection layer: 139 - type = 28
 Detection layer: 150 - type = 28
 Detection layer: 161 - type = 28
/mydrive/yolov4/test_images/tst1.jpg: Predicted in 32.886000 milli-seconds.
Raw_Apple: 100%
Raw_Apple: 100%
Ripe_Apple: 100%
Raw_Apple: 100%
Raw_Apple: 100%
Ripe_Apple: 100%
Raw_Apple: 100%
Unable to init server: Could not connect: Connection refused
```

figure 4. 23 Test result 4: clusters of apple

In figure 4.22 number of apples are detected according to their classes. The predictions finished within 32.88 milli-seconds.

4.6.2 Run apple detector over realSence camera and webcam - images

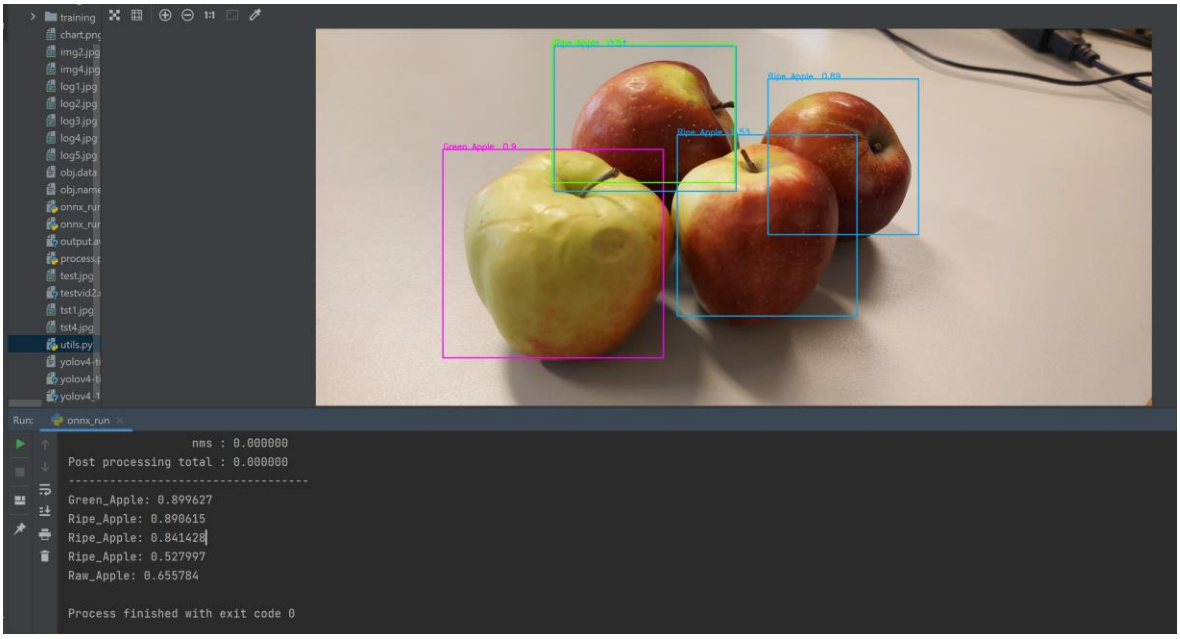
The Logitech C615 is connected to the computer to capture the image in different angles and lighting conditions and the test results are as follows;



```
Post processing total : 0.000000
-----------------------------------
Ripe_Apple: 0.985529
Ripe_Apple: 0.960798
Ripe_Apple: 0.800510
Raw_Apple: 0.696394

Process finished with exit code 0
```

figure 4. 24 webcam captured image with high saturation

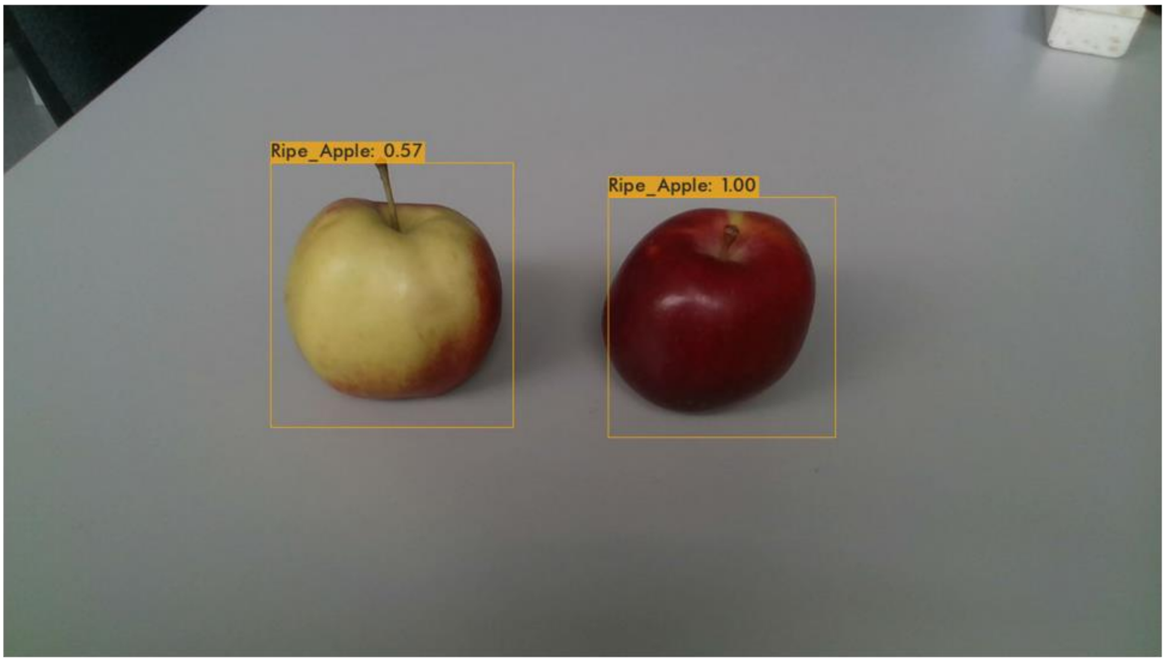


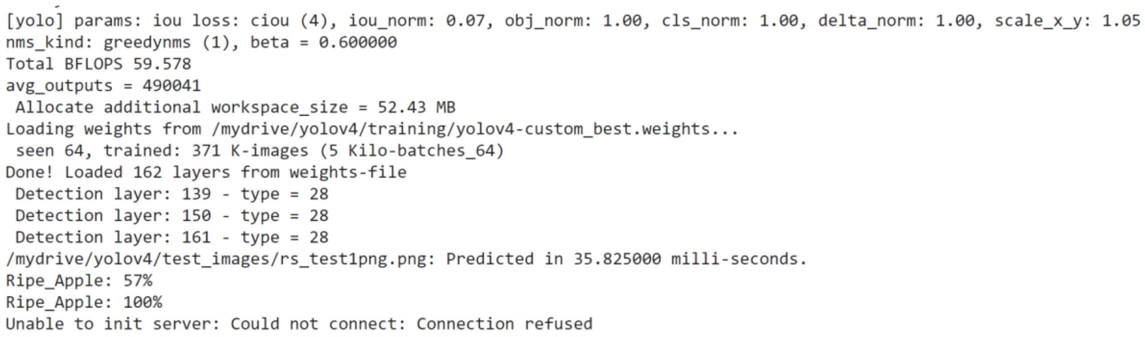
```
                nms : 0.000000
Post processing total : 0.000000
-----------------------------------
Green_Apple: 0.899627
Ripe_Apple: 0.890615
Ripe_Apple: 0.841428
Ripe_Apple: 0.527997
Raw_Apple: 0.655784

Process finished with exit code 0
```

figure 4. 25webcam captured image under shading condition

The images captured from realsence color camera are tested and verified the accuracy.





```
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05
nms_kind: greedynms (1), beta = 0.600000
Total BFLOPS 59.578
avg_outputs = 490041
 Allocate additional workspace_size = 52.43 MB
Loading weights from /mydrive/yolov4/training/yolov4-custom_best.weights...
 seen 64, trained: 371 K-images (5 Kilo-batches_64)
Done! Loaded 162 layers from weights-file
 Detection layer: 139 - type = 28
 Detection layer: 150 - type = 28
 Detection layer: 161 - type = 28
/mydrive/yolov4/test_images/rs_test1png.png: Predicted in 35.825000 milli-seconds.
Ripe_Apple: 57%
Ripe_Apple: 100%
Unable to init server: Could not connect: Connection refused
```

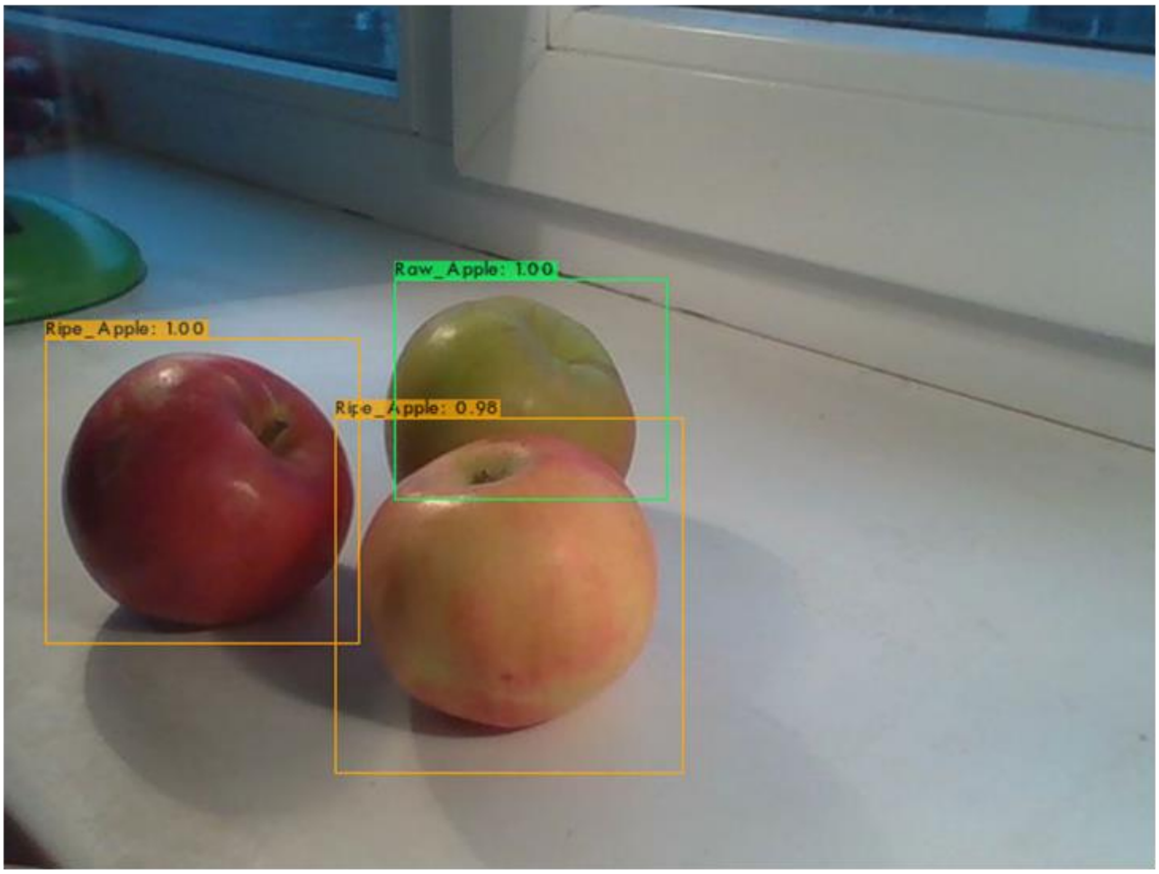
figure 4. 26 Tested on realsence image

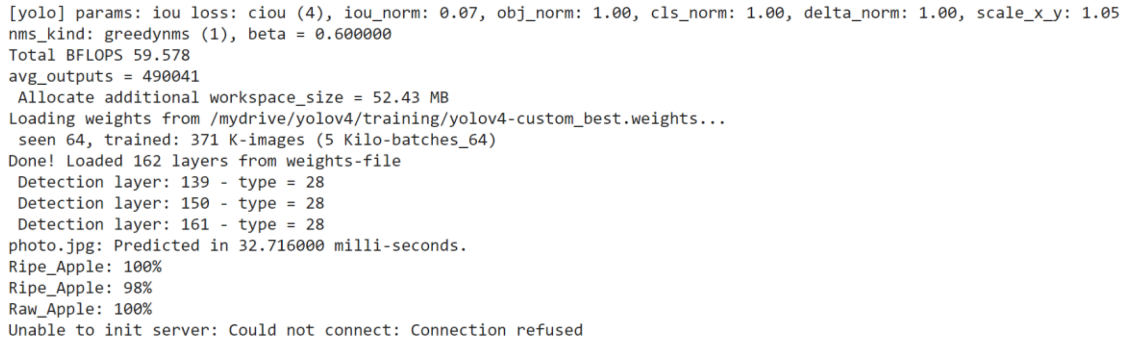
The RGB image tested using apple detector and the above figure 4.26 shows the apples detected with confidence score 57% and 100% for both apples within the class ripe apple.

To access the webcam, use code from Google Colab's Code Snippets, which has a number of helpful code functions for completing different tasks. For the purpose of this thesis, use the Camera Capture code snippet, which runs JavaScript code to use computer's webcam. The code snippet will take a webcam frame, which we will then use to detect objects using custom YOLOv4 model (given in the appendix). The code snippet provided by Google Colab for camera capture except for the last two lines which run the detector on the saved image that used in the previous command in the above section 4.6.1.

In the webcam part, a couple helper functions run this cell here is just converting the images from JavaScript objects into open CV and vice versa. because to actually utilized the webcam within Google colab workstation. running the yoloV4 on webcam images, have this take photos function that is just taken from the code snippets of camera capture here and customized. so this is the base code for this function right here in camera capture that just utilizes JavaScript code so implements JavaScript code in order to access the local machines webcam.

Once the JavaScript image is captured, then running the detections using dark net helper function that ran up above and then looping through the labels confidences and bounding boxes of each detection within detection and then drawing it on to the image itself and then just saving the image.





```
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05
nms_kind: greedynms (1), beta = 0.600000
Total BFLOPS 59.578
avg_outputs = 490041
 Allocate additional workspace_size = 52.43 MB
Loading weights from /mydrive/yolov4/training/yolov4-custom_best.weights...
 seen 64, trained: 371 K-images (5 Kilo-batches_64)
Done! Loaded 162 layers from weights-file
 Detection layer: 139 - type = 28
 Detection layer: 150 - type = 28
 Detection layer: 161 - type = 28
photo.jpg: Predicted in 32.716000 milli-seconds.
Ripe_Apple: 100%
Ripe_Apple: 98%
Raw_Apple: 100%
Unable to init server: Could not connect: Connection refused
```

figure 4. 27 Test run over web cam image

In the figure 4.23 consists of image and test results. Here 3 apples are detected such as two ripe apples with confidence score 100% and 98% respectively. Also detected one raw apple with 100% confidence score.

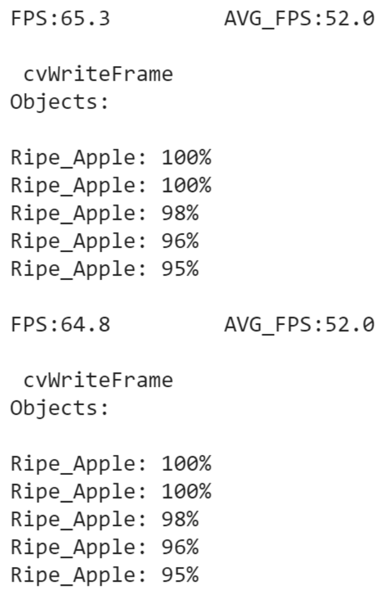
4.6.3 Run apple detector over video file

The process is same as that of section 4.6.1, upload the captured video file to the drive. After that run the following command (The threshold flag sets the minimum accuracy required for object detection).

```
# run your custom detector on a video with this command. This saves the output
video with the detections in the output path

!./darknet detector demo data/obj.data cfg/yolov4-
custom.cfg /mydrive/yolov4/training/yolov4-custom_best.weights -
dont_show /mydrive/yolov4/test_videos/test_vid1.mp4 -i 0 -
out_filename /mydrive/yolov4/test_videos/result_vid1.mp4 -thresh 0.5
```

This will give an output that will saved to the google drive. This used to check the accuracy and speed of detection.



```
FPS:65.3         AVG_FPS:52.0

 cvWriteFrame
Objects:

Ripe_Apple: 100%
Ripe_Apple: 100%
Ripe_Apple: 98%
Ripe_Apple: 96%
Ripe_Apple: 95%

FPS:64.8         AVG_FPS:52.0

 cvWriteFrame
Objects:

Ripe_Apple: 100%
Ripe_Apple: 100%
Ripe_Apple: 98%
Ripe_Apple: 96%
Ripe_Apple: 95%
```

figure 4. 28 Detection output

Figure 4.28 gives the detection over a video . The average frame rate obtained from the result was 52.0.fps that was a better frame rate for the apple detector model.

4.6.4 Run apple detector over a live stream

The method used was, open up a video stream add the input so constantly running a video stream running custom yolov4 for detections on each frame of the video because, yolov4 runs in milliseconds. So it will be possible. Then write custom bounding box of yolov4 onto a blank RGB image as an overlay. All the image has is the bounding boxes nothing else and then overlay that back onto the video stream and then it results in a real time yolov4 object detection. it's technically one frame behind because you're getting the detections on one frame and applying the overlay of those detections onto the next frame.

Darknet have a darknet.py file with several commands and functions within this python file. So for this test utilizing this in order to do the yolov4 detection. There's dark net helper function that just passes in the image the width and the height of the image. it is going to go ahead and run the detect image command from the darknet.py file on the image and get those detections. so it does all of the preprocessing of the image rescaling it and then gets a ratio for the bounding boxes and actually does the detections and then returns them.

in order to do the detection, run video stream, which is a JavaScript code, and run the helper function. open the video stream, and then do the detections on each frame so while true it's going to keep on looping through those detections and saving it into this blank array. the ‘bbox_array’ this is the transparent overlay. Then saving the rectangle bounding box and the text of the class onto the blank overlay and then it is just going to write it bbox update the overlay each frame then gets passed into the video frame. There is a little delay because of the one frame behind. In actual case it is negligible. Video frames are fed as input to yolov4.

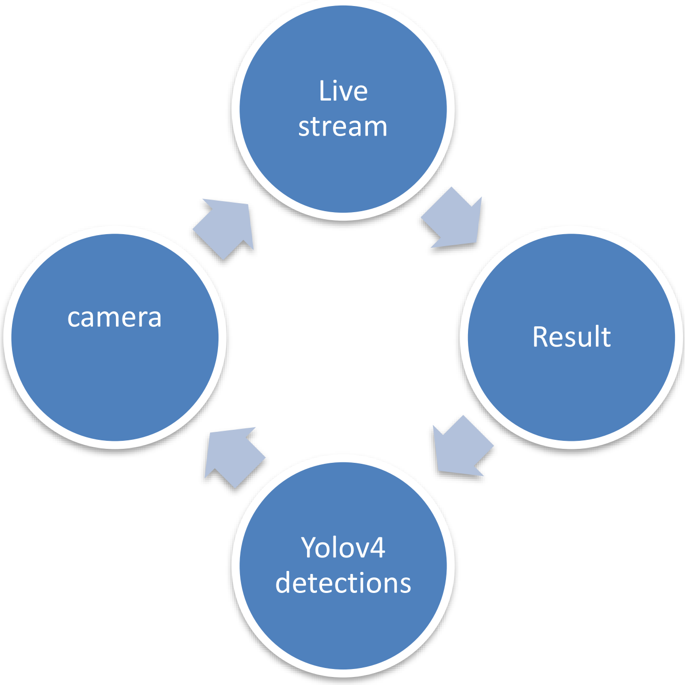


figure 4. 29 Process flow of live stream detection

With YoloV4, was able to achieve 25 FPS using GPU, with better precision that can be enhanced further. And, using the CPU, it was able to achieve a frame rate of roughly 7 frames per second. here used a 416 pixel image size. it is possible to increase the frame rate by lowering the image size by a factor of 32.

4.6.4 Exporting weights and deployement

The model weight files can be used for inference over new scenarios and images. To do so, point the model at the dataset’s test set, and point the detection script to the custom weights. The trained apple detector weight files are currently in the darknet format and this can be easily converted into other framework such as pytorch, Tensorflow, Tensorflow Lite, TensorRT. For the easiness of conversion, the apple detector model is converted to ONNX format. ONNX Runtime is a cross-platform inference and training machine-learning accelerator. ONNX is a model representation that allows to smoothly go from one environment to the next.

5 SUMMARY

The development of an apple detection and localization method using the YOLOv4 object detection model was detailed in this thesis. The proposed solution for detecting the apples to check whether it is ripe or not, using YOLO algorithm was trained and validated on RGB images. Custom datasets gathered from a complex environment helps overcome the limitations of deep learning. The data augmentation methods introduced in the YOLOv4 such as bag of freebies, Mish activation are reduce missed detections and inaccuracies and make the model more intelligent.

Three classes such as ripe apple, raw apple, and green apple were created. To test the developed model by using a realSence camera and C615 webcam has applied yolo for object detection and localization in an image. The algorithm divides the image into regions and predicts bounding boxes and probabilities for each region using a single neural network applied to the entire image. The estimated probabilities are used to weight these bounding boxes. following non-max suppression, which ensures that the object detection algorithm identifies each object only once. After those outputs the object with its bounding box and confidence score.

Although the time took to training the model was greater than expected. The saved weights are utilized for testing. The performance of the model was verified by checking the mean average precision of each weight file. The mAP for the yolov4-custom_6000.weight file is 100% and the average IoU is 89.10%. class prediction for green, ripe and raw apple are 100%. The average loss after 6000 iterations was 0.130, which’s good for this model to identify the fruit. . The mAP for the yolov4-custom_best.weight file is 85.81% and the average IoU is 63.24%. The probabilities of each class obtained from the best weights are 84.83%, 86.98%, and 85.62% for green apple, ripe apple, and raw apple respectively. From the experimental results it is clear that the apple detector model outperformed all the other traditional detection models with respect to the average precision. YOLO based apple detector achieved 0.98, 1, 0.99 on precision, recall and F1 score on apple detection.

Test results showed that the proposed method could accurately identify the object extremely faster than previous methods. The average time required for prediction was estimated at 32 milli-seconds on a Tesla T4 48 milli-seconds on RTX 2080 super. The major challenges of object detection addressed at the beginning of the work like overlapping, clustering, and occlusions were resolved.

5.1 Limitations

The google colab GPU is free but sometimes it shows errors like connection errors, the runtime just fails to work from time to time. Only accessible for 12 hours in a row. Somehow this might be influencing the training, it may cause intermittent breaks while training. It takes hours to finish the process.

Generating the image dataset is a tedious task. There are hundreds of verities of apples out there. Each of them has different criteria for predicting whether they are ripe or not. So it needs to create a dataset separately for each one for better accuracy. This work only considered the Paula red, Gala specious, and green apples. This will hopefully work for red, lady Sugarbee, and Granny Smith as well.

Check the apple detector outdoor is not possible right now. Because in Estonia, apple harvesting has done in August. An indoor test has been done to validate the model.

The varying lighting conditions are a concern here. The algorithm has some limitations to the higher exposure. The configuration has the exposure, saturation, and hue parameters. In some cases, a higher exposed image gives false detections. This may create shadows over the fruits.

5.2 Future work

The detector developed in the thesis has been tested in laboratory indoor conditions. The next step is to test the practical application. Then verify the precision of the model.

There is some other deep learning framework implement YOLO such as TensorFlow and PyTorch. This framework requires dependencies to implement yolo. The same detector model can be built in this framework to check the possible outcomes.

The developed model mainly focused on day-time harvesting purposes. A suitable night vision can be integrated into the detector for all-weather conditions. The lighting system and illumination factor should be considered.

The further step will be merging the vision and manipulation part altogether to evaluate the accuracy of harvesting. Export the object coordinates to the manipulator and lead the end effector to harvest the fruit. Combine the whole harvesting time for the model.

On a single farm, 10-15 breeds of apple will be there. Every one of them has different criteria to predict ripeness. Each data can be collected and train the model according to the custom data and use the weights for detection and improve the vision algorithm in terms of accuracy and speed. Instance segmentation and grasping estimation process have to be considered in the future work for further development of the project.

Same model data can be used for training the model using YOLOv4-Tiny, which is the smaller version of YOLOv4. But, emphasize model speed and and a smaller model size for inference even on a smaller hardware like Jetson nano and RasPi.

5 KOKKUVÕTE

Käesolevas lõputöös kirjeldati üksikasjalikult õunte tuvastamise ja lokaliseerimise meetodi väljatöötamist, kasutades YOLOv4 objektide tuvastamise mudelit. Väljapakutud lahendus õunte tuvastamiseks, et kontrollida, kas need on küpsed või mitte, kasutades YOLO algoritmi, treeniti ja valideeriti RGB-piltidel. Komplekssest keskkonnast kogutud kohandatud andmekogumid aitavad ületada süvaõppe piiranguid. YOLOv4-s kasutusele võetud andmete suurendamise meetodid, nagu bag of freebies, Mish'i aktiveerimine, vähendavad vahelejäänud tuvastusi ja ebatäpsusi ning muudavad mudeli intelligentsemaks.

Loodi kolm klassi, nagu küps õun, toores õun ja roheline õun. Testida väljatöötatud mudelit, kasutades realSence'i kaamerat ja C615 veebikaamerat on rakendanud yolo objekti tuvastamiseks ja lokaliseerimiseks pildil. Algoritm jagab pildi piirkondadeks ja ennustab iga piirkonna jaoks piiritlevad kastid ja tõenäosused, kasutades kogu pildi suhtes rakendatud ühte neuronivõrku. Hinnangulisi tõenäosusi kasutatakse nende piiritlevate kastide kaalumiseks. järgides mitte-maksimumtasakaalu, mis tagab, et objekti tuvastamise algoritm tuvastab iga objekti ainult üks kord. Pärast seda väljastatakse objekt koos selle piiritletud kasti ja usaldusskooriga.

Kuigi mudeli treenimiseks kulunud aeg oli oodatust suurem. Salvestatud kaalusid kasutatakse testimiseks. Mudeli toimivust kontrolliti iga kaalufaili keskmise täpsuse kontrollimisega. Yolov4-custom_6000.weight faili mAP on 100% ja keskmine IoU on 89,10%. rohelise, küpse ja toore õuna klassiprognoos on 100%. Keskmine kaotus pärast 6000 iteratsiooni oli 0,130, mis on hea selle mudeli jaoks puuvilja tuvastamiseks. . mAP faili yolov4-custom_best.weight jaoks on 85,81% ja keskmine IoU on 63,24%. Parimate kaalude põhjal saadud iga klassi tõenäosused on vastavalt 84,83%, 86,98% ja 85,62% rohelise õuna, küpse õuna ja toore õuna puhul. Katsetulemustest selgub, et õuna tuvastamise mudel on keskmise täpsuse poolest parem kui kõik teised traditsioonilised tuvastamismudelid. YOLO-l põhinev õunadetektor saavutas õunte tuvastamisel täpsuse, tagasikutsumise ja F1-skoori 0,98, 1, 0,99.

Katsetulemused näitasid, et kavandatud meetod suudab objekti väga täpselt tuvastada äärmiselt kiiremini kui varasemad meetodid. Prognoosimiseks kuluvaks keskmiseks ajaks hinnati 32 milli sekundit Tesla T4-l 48 milli sekundit RTX 2080 superil. Töö alguses käsitletud peamised objektide tuvastamise probleemid, nagu kattuvus, klasterdamine ja oklusioonid, said lahendatud.

5.1 Piirangud

Google Colab GPU on tasuta, kuid mõnikord näitab see vigu, nagu näiteks ühendusvead, aeg-ajalt lihtsalt ei tööta. Kättesaadav ainult 12 tundi järjest. Kuidagi võib see mõjutada treeningut, see võib põhjustada treeningu ajal aeg-ajalt katkestusi. Protsessi lõpetamine võtab tunde.

Pildi andmestiku genereerimine on tüütu ülesanne. Seal on sadu erinevaid õunu. Igaühel neist on erinevad kriteeriumid, et ennustada, kas nad on küpsed või mitte. Seega tuleb parema täpsuse saavutamiseks luua iga õuna jaoks eraldi andmekogum. Käesolevas töös vaadeldi ainult Paula punaseid, Gala erilisi ja rohelisi õunu. Loodetavasti töötab see ka Red Delicious, Pink Lady Sugarbee ja Granny Smithi puhul.

Kontrollida õunade detektor väljas ei ole praegu võimalik. Sest Eestis on õunakoristus tehtud augustis. Mudeli valideerimiseks on tehtud siseruumides test.

Siinkohal on probleemiks erinevad valgustingimused. Algoritmil on mõned piirangud suurema valgustuse suhtes. Konfiguratsioonil on ekspositsiooni-, küllastus- ja värvitooni parameetrid. Mõnel juhul annab kõrgema valgustusega pilt valedetektorid. See võib tekitada varjud puuviljade kohale.

5.2 Tulevased tööd

Töös välja töötatud detektorit on katsetatud laboratooriumi sisetingimustes. Järgmise sammuna on vaja katsetada praktilist rakendust. Seejärel kontrollida mudeli täpsust.

On mõned teised süvaõppe raamistikud, mis rakendavad YOLO-d, nagu TensorFlow ja PyTorch. See raamistik nõuab Yolo rakendamiseks sõltuvusi. Sama detektorimudelit saab ehitada selles raamistikus, et kontrollida võimalikke tulemusi.

Välja töötatud mudel keskendus peamiselt päevase saagikoristuse eesmärkidele. Sobiva öise nägemise saab integreerida detektorisse kõikvõimalike ilmastikutingimuste jaoks. Arvesse tuleks võtta valgustussüsteemi ja valgustustegurit.

Edasine samm on nägemise ja manipuleerimise osa ühendamine kokku, et hinnata korje täpsust. Objekti koordinaatide eksportimine manipulaatorisse ja lõpptoimija juhtimine vilja koristamiseks. Ühendage kogu koristusaeg mudeli jaoks.

Ühes põllumajandusettevõttes on 10-15 õunasorti. Igaühel neist on erinevad kriteeriumid küpsuse prognoosimiseks. Iga andmeid saab koguda ja treenida mudelit vastavalt kohandatud andmetele ning kasutada kaalusid tuvastamiseks ja parandada nägemisalgoritmi täpsuse ja kiiruse osas. Projekti edasiseks arendamiseks tuleb tulevases töös kaaluda instantside segmenteerimist ja haaramise hindamisprotsessi.

Sama mudeli andmeid saab kasutada mudeli treenimiseks, kasutades YOLOv4-Tiny, mis on YOLOv4 väiksem versioon. Kuid rõhutada mudeli kiirust ja ja väiksemat mudeli suurust järelduste tegemiseks isegi väiksemal riistvaral, nagu Jetson nano ja RasPi.

References

[1] F. D, "AI in Agriculture-present applications and impact," Emerj; The AI research and advisory company, 2020.

[2] Bureau of labor statistics, "Occupational outlook handbook, Agricultural workers," U.S Department of labor, USA, 2020.

[3] B. J, "Future work at the human technology frontier," National Science Foundation, 2020.

[4] B. B.S, "Robotic agriculture: The future of agricultural mechanisation?," in *European confrence on precision agriculture* , 2005.

[5] G. A, A. S, K. M, Z. j. Q and L. K , "Sensors and systems for fruit detection and localization: A review," *Computers and electronics in agriculture,* vol. 116, pp. 8-19, 2015.

[6] B. D, K. T, O. j. Y and H. T, "A Segentation algorithm for the automatic recognition of fuji apples at harvest," *AI- Autotion and emerging technologies,* vol. 4, no. 83, pp. 405-412, 2002.

[7] W. J, Z. D, F. C, B. X, Y. j. Z and J. W, "Automatic recognition vision system guided for apple harvesting robot," *computers & Electrical Engineering ,* vol. 5, no. 38, pp. 1186-1195, 2012.

[8] L. R, C. j. O and N. A, "Determination of the number of green apples in RGB images recorded in orchards," *Computers and Electronics in Agriculture,* no. 81, pp. 45-57, 2012.

[9] P. A, W. K, S. j. P and J. D, "Estimating mango crop yield using iage analysis using fruit at stone hardening and night time imaging," *Computers and electronics in Agriculture,* no. 100, pp. 160-167, 2014.

[10] S. A, D. J, K. M.R , M. C, Z. j. Q and L. K, "Design, Integration and field evaluation of a robotic apple," *Field Robotics,* vol. 6, no. 34, pp. 1140-1159, 2017.

[11] B. E, M. R, J. O and S. Ben, "Color-agnostic shape based 3D fruit detection," *Robotic Agriculture ,* vol. 1, no. 15, pp. 57-70, 2016.

[12] K. F, L. j. W and V. A, "Immature peach detection in color images acquired in natural illumination conditions using statistical classifiers and neural network," *Precision Agriculture,* vol. 1, no. 15, pp. 57-79, 2014.

[13] C. O, L. R and N. Ja A , "Estimation of the number of apples in color iages recorded in orchards," in *International conference on computer and computing technologies in agriculture* , springer, Berlin, 2010.

[14] Q. L, J. C, L. j. D and Y. Z, "Identification of fruit and branch in natural scenes for citrus harvesting robot using machine vision and support vector machine," *International Journal for agriculture-Biological engineering,* vol. 2, no. 7, pp. 115-121, 2014.

[15] J. L, E. J. S and T. D, "Fully convolutional networks for sematic segmentation," in *IEEE confrence on computer vision and pattern recognition (CVPR)* , Boston, MA, USA, 2015.

[16] S. Z, S. J, B. R, V. V, Z. S, D. j. D and C. H, "Conditional random fields as recurrent neural networks," *Computer vision and pattern recognition (cs.CV),* vol. 1, pp. 1529-1537, 2015.

[17] L. W, A. D, E. D, S. C, R. S, F. j. C-Y and B. A, "SSD: Singleshot multibx detetor," in *European conference on computer vision* , Springer, Berlin, 2016.

[18] Y. O, T. Y, H. K, T. F, H. j. A and A. I, "An automated fruit harvesting robot by using deep learning," no. 13, 2019.

[19] B. Alexey, W. Chien-Yao and M. L. Hong-Yuan, "YOLOv4: Optimal speed and accuracy of object detection," 2020.

[20] C. Wakamiya, "Object detection with yolo," DATA X, 2020.

[21] A. NG, "Google.com," 2018. [Online].

[22] P. Sharma, "www.analyticsvidhya.com," 4 November 2018. [Online]. Available: https://www.analyticsvidhya.com/blog/2018/11/implementation-faster-r-cnn-python-object-detection/?utm_source=blog&utm_medium=a-step-by-step-introduction-to-the-basic-object-detection-algorithms-part-1.

[23] K. Hanwen , Z. Hongyu, W. Xing and C. Chao, "Real-Time Fruit Recognition and Grasping Estimation for Robotic Apple Harvesting," *Sensors and Robotics,* vol. 20, no. 19, 2020.

[24] R. Joseph, D. Santhosh, G. Ross and F. Ali, "You Only Look Once: Unified, Real-Time Object Detection," *Facebook AI Research,* 2016.

[25] R. Joseph and F. Ali, "YOLO9000 Better, Faster, Stronger," Universty of Washington, Allen Institute for AI, 2016.

[26] R. Joseph and F. Ali, "YOLOv3: An Incremetal Improvement," Universty of Washington, 2018.

[27] R. Joseph and F. Ali, "YOLOv3: An Incremental Improvement," University of Washington, Washington, USA, 2019.

[28] H. Zhanchao and W. Jianlin, "C-SPP-YOLO: Dense Connection and Spatial Pyramid," College of Information Science and Technology, Beijing University of Chemical Technology, Beijing, China, 2019.

[29] S. Wan and S. Goudos , "Faster R-CNN for multi-class fruit detection using a robotic vision system.," *comput. Netw,* p. 168, 2020.

[30] c. Wenkang, L. l. Shenglian, L. Binghao, L. Guo and Q. Tingting, "Detecting Citrus in Orchard Environment by Using Improved YOLOv4," *Scientific Programming,* vol. 2020, p. 13, 2020.

[31] AlwxeyAB, "www.github.com," [Online]. Available: https://github.com/AlexeyAB/darknet.

[32] Intel, "https://www.intelrealsense.com/," Intel, [Online]. Available: https://www.intelrealsense.com/depth-camera-d415/.

APPENDICES

The Codes used in the project data files test results everything available in the following github repository

https://github.com/jesil94/apple-harvesting-robot-vision-system