CALIFORNIA STATE UNIVERSITY SAN MARCOS

PROJECT SIGNATURE PAGE

PROJECT SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

PROJECT TITLE: Fruits And Vegetable Detection For POS With Deep Learning

AUTHOR: Kavan Patel

DATE OF SUCCESSFUL DEFENSE: 12/02/2020

THE PROJECT HAS BEEN ACCEPTED BY THE PROJECT COMMITTEE IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE.

Dr. Xin Ye	Xin Ye	12/02/2020
PROJECT COMMITTEE CHAIR	SIGNATURE	DATE
Dr. Yanyan Li	[illegible]	12/02/2020
PROJECT COMMITTEE MEMBER	SIGNATURE	DATE
Name of Committee Member		
PROJECT COMMITTEE MEMBER	SIGNATURE	DATE

Fruits & Vegetable Detection POS with Deep Learning

Project Report

patel088@cougars.csusm.edu

Computer Science

California State University, San Marcos

Table of Contents

Abstracts

The self-checkout portal at the supermarket gives the idea to the classification of fresh fruits and vegetables. Nowadays, more self-checkout portals are added to the time saving of the customers at the supermarket. When it comes to fresh fruits and vegetables, It still needs to enter manually into the computer for purchase and, it is a bit time-consuming and increases cheating (By putting the wrong item name). We can make use of the camera at self-checkout to get a prediction of the items using machine learning. For example, when we put tomatoes on the counter, it detects the tomatoes through a semi-transparent bag and gives various tomatoes as a list. The problem with different object detection models is to see through semi-transparent bags to classify the image. You Only Look Once (YOLO) object detection did this job well If we train the model correctly. The main stages of Object detection are data acquisition, Augmentation, Model training, Model Evaluation, and Deployment. It gives 99.4% max accuracy on the training and testing dataset to classify 14 different classes for fruits and vegetables. On real-life images, it provides approx 90% accuracy on images to classify. Prediction execution performs under a sec is considered a good result for the self-checkout terminal.

List of Abbreviation

YOLO: You Only Look Once (Object Detection Model)

API: Application Programming Interface

AI: Artificial Intelligence

ML: Machine Learning

OpenCV: Open Computer Vision

FPS: Frame Per Second

R-CNN: Regional Based Convolution Neural network

TFlite: TensorFlow Lite

TF: TensorFlow

GUI: Graphical User Interface

DIY: Do It Yourself

GPU: Graphical Processing Unit

IoT: Internet of Things

1. Introduction

Machine Learning is more and more popular nowadays. It is implemented in most places, either mobile technology or real-life software (such as YouTube Recommendation, Robotics Object detection). There are still so many places where we can apply machine learning and make existing things more productively. One of my observations is the Self-checkout station at grocery marketplace like Kroger, Walmart. Nowadays, supermarkets are adding more and more self-checkout stations to save the time of customers. Self-checkout stations are goods for the items which have a barcode on them. When we talk about fruits and vegetables, it needs to be fresh, so, it sells as loose. Adding machine learning on an existing camera at the self-checkout station can solve the problem. With the Application of machine learning, the camera classifies items into a correct category (e.g., grapes, lemon, banana, etc.) So, customers no need to search for items by typing in software.

Let's Discuss Problem with the existing system.

1) **Quality Prevention**

   According to the United Nations, roughly one-third of the food produced in the world for human consumption gets lost or wasted every year [1]. By bagging fruits and vegetables and barcode them, it can save time at a self-checkout station, but it can be one of the reasons for the food west. By bagging food, it can increase the chances to deteriorate food quality faster than the normal time. Machine learning can classify the fruits and vegetables in the correct category, and auto-scaling can solve this problem and prevent the quality of the food.

2) **Cheating Protection**

   Sometimes some customers intentionally or accidentally type different fruit or vegetable at the self-checkout portal, for example, putting avocado on the counter they typed tomato in the system which is cheaper, and they pay less for it [2]. For items with the barcode, some people damage it or replace it with the cheaper items that cannot be detected at the self-checkout [3]. By the use of machine learning, this problem can be solved, and it can save a big chunk of money for supermarkets.

For developing a Proposed system, the main issue is to detect the object through the semi-transparent bag. There are existing models available to create custom object detection like Regional Based Convolution Neural Network (R-CNN), You Only Look Once (YOLO), Faster R-CNN object detection models. From them, the YOLO object detection model is working well with this problem as it is fast, reliable, and flexible to work with a custom dataset. As there is Tensorflow-Yolov4-Tflite API is available to use so, that makes our product deploy locally without using the cloud and, there will be no extra cost to manage cloud services. For security purposes, the cameras are available and, we can make use of those cameras.

1.1. Contribution to This Project

- Creation of great dataset Which consists of 14 different classes. Approximately 656 images taken by phone
- Data Labeling: Manually label all 656 images for different classes.
- Data Augmentation: There are multiple APIs available for data augmentation, But Data augmentation with the labels associated with it, is a challenging task to find API.
- Model Training with using YOLOv4 Custom Object Detector
- Converting model with TensorFlow-YOLOv4-Tflite API and writing script for Deploy converted model on GUI.

•

2. Technologies and Related work

2.1. Technologies: Introduction of machine learning model in use

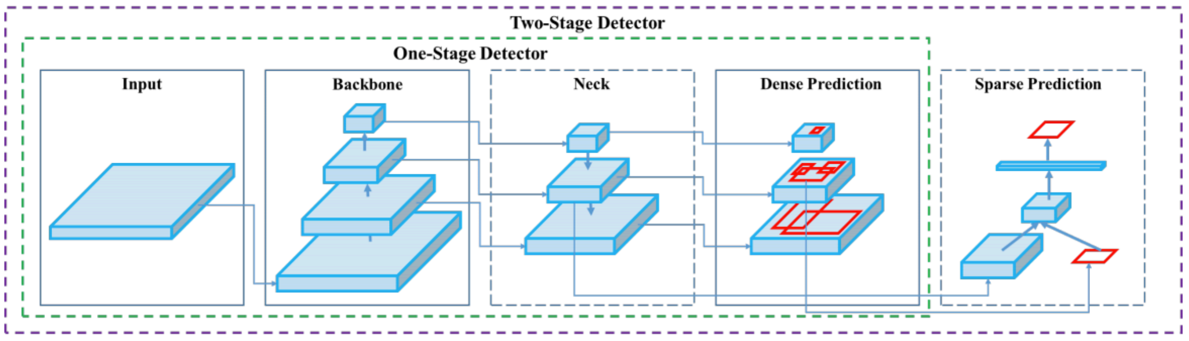
There are Different Object detection models available to use like R-CNN, YOLO, Faster R-CNN, etc.

Among them, the YOLO object detection model is giving more accuracy with great speed as it uses Single-shot detection for speed and two-shot detection for better accuracy. For better speed YOLO object detection model is a good fit for the proposed project.

YOLO object detection

You only look once (YOLO) is a state-of-the-art, real-time object detection system [4]. YOLO object detection model is very well known for its speed, accuracy also flexibility. The first three YOLO versions have been released in 2016, 2017, and 2018 respectively [5]. YOLOv4 is related to it's previous YOLO version. It was released in early 2020. YOLOv5 was introduced in 2020 after a few months of YOLOv4 but, it is controversial and, there is no such difference between YOLOv4 and YOLOv5. Here, I worked YOLO v4 for this project as it is mature enough and resources are available to do some extra work.

YOLO v4



Input: { Image, Patches, Image Pyramid, … }

Backbone: { VGG16 [68], ResNet-50 [26], ResNeXt-101 [86], Darknet53 [63], … }

Neck: { FPN [44], PANet [49], Bi-FPN [77], … }

Head:

Dense Prediction: { RPN [64], YOLO [61, 62, 63], SSD [50], RetinaNet [45], FCOS [78], … }

Sparse Prediction: { Faster R-CNN [64], R-FCN [9], … }

Figure 2.1.1 YOLO v4 Object detection Model [6].

The YOLO v4 Object detection model is based on its prior object detection model, which is YOLOv3.

Layers

Input: Here, it takes Input as an image.

Backbone: YOLO uses different convolution neural networks as a backbone for ex VGG16 for classification, 50 layers of ResNet neural network, etc.

Neck: This layer used to predict object on a different scale by using future pyramid networks like FPN, Bi-FPN

Head: This layer is used to predict the dense area where two objects are very close and, for that Single-Shot Detector (SSD), Region Proposal Network (RPN), etc.

For the objects are on some distance in those case for the prediction Faster R-CNN used for prediction in YOLOv4. The sparse prediction uses a regional method for detection, so it also considers as second shot detection.

2.2. Related Work:

Making Custom Data set and Training Object detection is still new and, It is not a mature thing. But the use case of this technology is limitless.
For specific context related to self-checkout stations, The work done by Fruits and Vegetable Classification from the live video [7] Lukas Danev. They Make a micro-app and Make a custom dataset which separates by with bag and without the bag. And with Segmentation and Gaussian Bayes Classifier, They get good accuracy on their model 85% for the top three predictions, Which is great at those time 2017.
Another study is Tunnel type Image-based system [8]. Which is considered as the same context, but it includes hardware to implement.
Tiliter vision [9] They do the same context, where they make a self-checkout machine which takes an image as input and scale it and give the price for that. Their proposed system works well with the fruits and detection without a bag. It cannot show bag detection. It is not the optimal solution, to buy fruits or vegetables without a bag if we buy it in bulk.
Another Study Conduct on the topic Simplifying Grocery shopping with Deep Learning [10] by Jing Ning, Yongfeng Li, Ajay. Ramesh Where they mask image labels by pixel-level annotation (It fills out the image by the border of an object). They use the R-CNN model for the training and achieved 84% of mAP(mean average precision) for a default threshold of 50%. Their main focus on all items of the grocery store that includes items with the barcode as well.
The Research-based on fruit verity classification is similar research as they classify fruits and their verity as well. For the classification problem, they used a double-track method where they used two 9-layer CNN one is used for the RoI detection and the other is used for classifying fruit category with the detected RoI. They got a good classification speed which is an avg 350ms, It is slightly more than the YOLOv3 Realtime object detection [11].

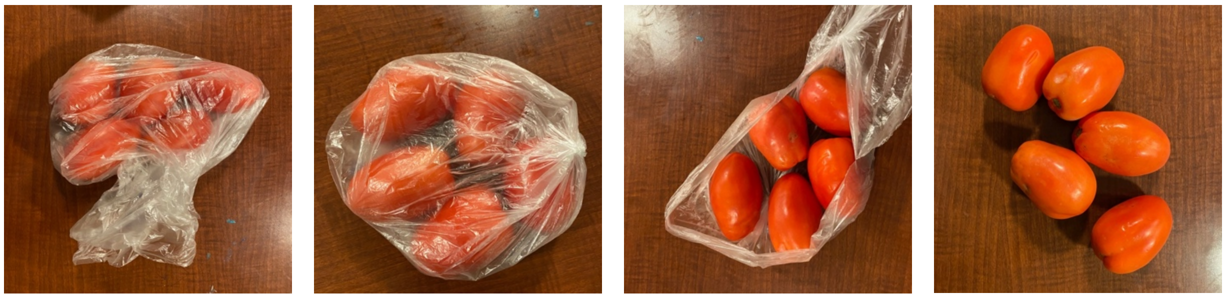
3. Dataset Acquisition

The most important part of the Dataset Acquisition is how to capture the images, which can give you a good result after model training. We start from the basic where we make small datasets through the big datasets. All of the images were captured, With Default settings in the camera of iPhone 11.

- To train an object detection model, the datasets play an important role. There are different datasets available on Kaggle, Open Image Dataset By google, and Some other websites.
- During the initial stage of my project, I reviewed many data set from them Fruit 360 one of them, but the limitation of that dataset is it is specifically developed to reticular fruit detection by segmented portion.
- For the YOLO object detection, we need to label a bunch of images with the background.
- Although Fruit 360 dataset works with R-CNN as per research, it will struggle with seeing through the semi-transparent plastic bag, which is the most crucial part of this project.
- To Conclude All this Scenario, I decided to make a custom dataset for my custom model.

Iteration 1:

- First Iteration contains only 1 class with 69 photos of the tomatoes with a bag and without a bag, which is such a small dataset for any object detection model.
- It is developed for the experimental purpose for checks how it can behave with the YOLO object detection model.



3.1 Iteration 1: Tomato with bag & without bag

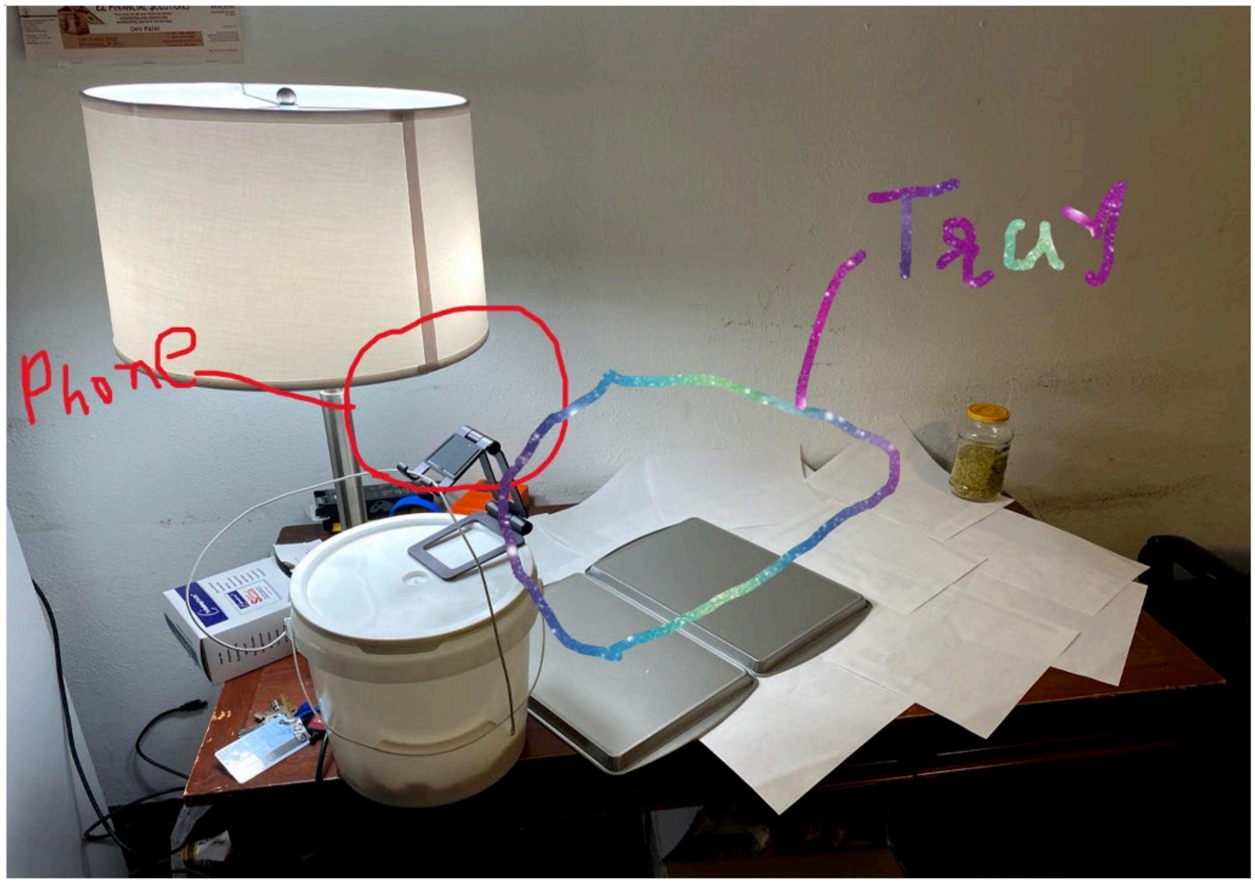
Iteration 2:

- The second iteration contains three different classes, which include lemons, chilies, apples. These classes are with and without a bag.
- Here, there were 165 images taken during the 2nd iteration. It is randomly captured images without worrying about the background and other things.

3.2 Iteration 2: 3 Class datasets with & without bag

Iteration 3:

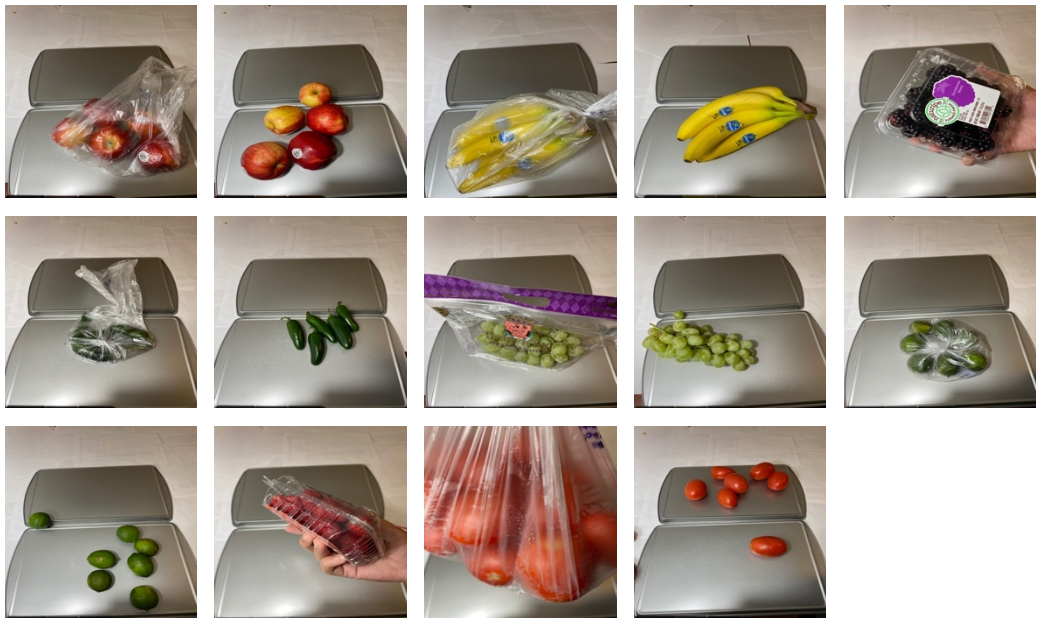
- During the 3rd Iteration, we got exposure to dataset creation. Here we decided to make a platform that depicts the environment at the self-checkout stations.
- To depict the same environment as self-checkout stations, we put the chrome plate as a background, and for steadiness in all photos, we set up the phone in one place until we got the entire dataset.
- Here is a little look at the DIY-home setup of our platform.



3.3 Setup for Data Acquisition

- Here, during the 3rd iteration, we create 14 separate classes in which we took approx 50 images of every individual class.
- 14 class are banana-bag, banana, blackberries, raspberry, lemon-bag, lemon, grapes-bag, grapes, tomato-bag, tomato, apple-bag, apple, chili-bag, chili
- For proper lighting, we set up a lamp backside of the phone stand.

•



3.4 Iteration 3: 14 Classes with & without bag

- Here, we took photos by putting an object near and far to the camera. That way our model can learn for all of the scenarios.
- Here, after Iteration 3, we took a total number of 656 images with a different angle and different distance from the camera.
- To give a feel like a self-checkout station we put chrome plate so when it will get to the production model have a little more accuracy due to the identical background.
- In some of the images, we intentionally keep our hand in the images. It will give the feel of a customer's hand during the detection of the item.
- Some of the images we keep half in the bag half outside of the bag. That way we can assume if some of the items keep outside of the bag then it can still perform the detection.

•

4. Fruits & Vegetable Detection Implementation

The Models

YOLOv4 Custom Object Detection Model with Custom Dataset.

The custom dataset consist total of 4592 images of 14 different class from which,

16.48% are testing images: 650

83.52% are training images: 3942

To feed the model.

Environments

Code implementation is implemented on local hosts and cloud as well.

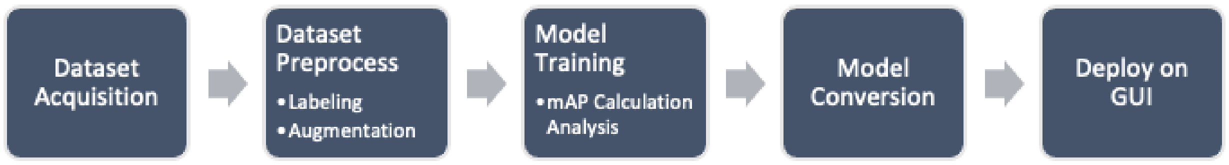
Cloud Environment: To training the model, it needs too much computing power that, we used Google's Colab. It provides excellent GPUs like Tesla k8 up to 12 Gb and, it provides a great IDE environment to write the python code.

Local Environment: After the training model, it does not require GPU anymore so, All the code for model conversation and GUI development is done locally by using python 3.8 and Jupiter Notebook.

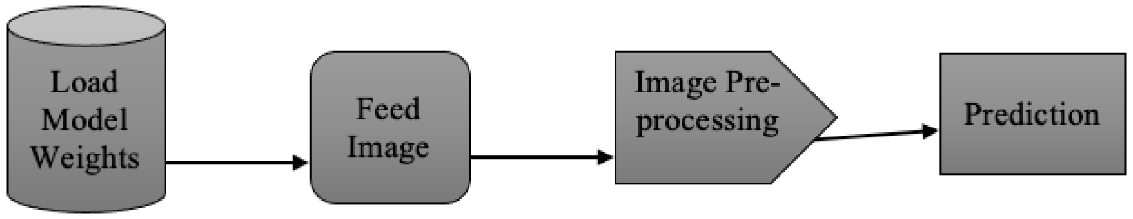
Goals

The goal is to develop a good model, which will be converting to format that we can deploy it locally. By using its weight, we can be able to perform predictions.

4.1. Development Process



Architecture



4.1.1. Image Labeling

- With the dataset labels for every image, the deep learning model can recognize an object and learn a particular object for a certain class.
- There are plenty of Image labeling tools available e.g., LabelImg, VGG Image Annotation, Labelbox.
- For the YOLO object detection, each label needs to be labelled an object in a rectangular shape on its class. LabelImg is the best suitable, popular, and easily available python tool.
- We manually labeled 656 images, which generate 656 .xml label files for each image.

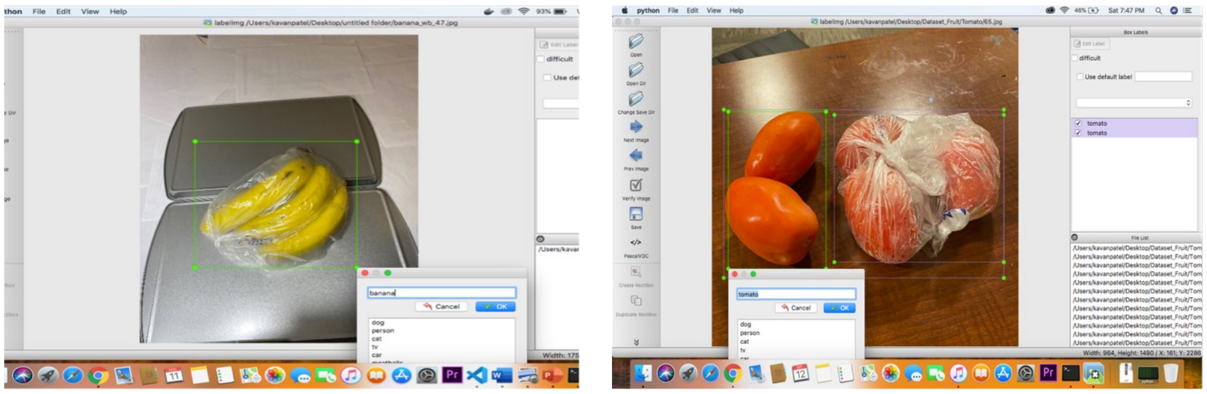


Figure 4.1.1 LabelImg Interface

4.1.2. Data Augmentation

- Data Augmentation is a step to increase the size of the data set by doing some transformation on the images for e.g., Rotating an image by some angle, adding noise to an image, shifting image, mirroring the image, Adding filters to the image, etc.
- For Image augmentation, there are plenty of resources are available but image augmentation with Yolo labels CloDSA is one of the base open-source programs with using that we can do our necessary transformation to the image as well as labels [12]. Here, we used a notebook provided by CloDSA to Do augmentation on our dataset.
- CloDSA accepts image files and .txt formatted labels. During data labeling, we created labels in the .xml file. We used XmlToTxt Project by Isabek to convert all .xml file to .txt file [13].
- For our data augmentation, we do not use all transformations of CloDSA for e.g., adding filters can change the color of the fruits or vegetable that can lead to our model in a different direction. So, we used Vertical flip, Horizontal flip, 90-degree rotation, average blurring, Raise Hue, Horizontal and vertical flip, and keep one copy of the original image.
- Here we did 6 types of transformation, so our dataset size increased from 656 to 4592 images. If we consider storage it increases from approx. 5Gb to 15Gb.

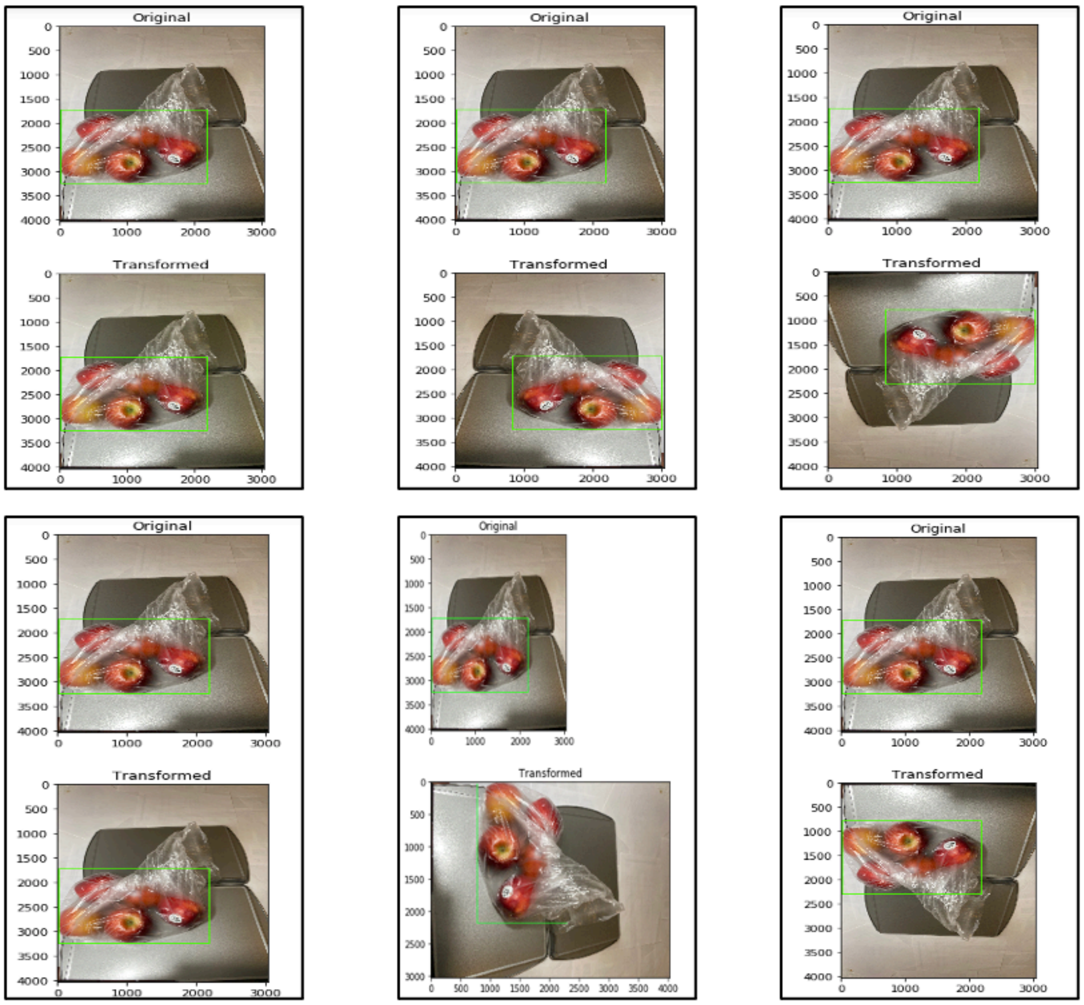


Figure 4.1.2 Various Transformation on the Image [13]

4.1.3. Model Training

The ultimate goal for the training model is to get a good result with minimum loss. To train any Deep Learning model requires powerful GPU Hardware. Here we are training the YOLOv4 Custom object detection model. As this model deals with the images instead of only variables powerful GPU are necessary. Google Colab already provides a free Powerful GPU to use which is Tesla K8 up to 12Gb. For that, we move all our datasets to Google drive to use them.

These are the Steps involved to Training to get the actual model:

Cloning & Building Darknet on Colab Virtual machine:

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 [14].

Building Darknet on our virtual environment is in just two steps

1) Clone darknet repository from git hub [15].
2) Use !make command to build it.

1. **Mounting Google drive with Colab:**

   We are used google drive as a storage for our dataset. The mounting drive is just a need drive.mount('/content/gdrive') command.

   Before the training, we move our custom dataset stored on google drive, to google Colab's virtual machine. The reason to use google drive is to move the data to Colab with using the drive is too fast.

2. **Attach Configuration files for training:**

   Cfg file is attached to tell the model that what is the max batch size, step size, number of classes we are training for, filters, etc.

   We got a sample yolov4-custom.cfg file from the darknet repository of AlexeyAB [15].

3. **Downloading yolov4 weights to make training process faster:**

   By downloading pre-trained yolov4 weights makes the training process more accurate and faster.

   It can use their pre-trained model's weight and filter in our training, which helps to learn our model faster in a short period.

4. **Training:**

   Yolov4 custom object detection traing is started by following line.

   !./darknet detector train data/obj.data cfg/yolov4-obj.cfg yolov4.conv.137 -dont_show -map[15].

   Where first it selects train command of detector class from darknet environment.

   Then we attached obj.data file which describes different paths:

   The number of classes, training & validation files path in .txt format, a path for obj.names, a path for the backup folder.

Here obj.names file describe all class names, in our case banana, apple, banana-bag, chili, etc.

4.1.4. mAP (Mean Average Precision) Calculation

the mAP is the most common term in deep learning to evaluate models. It calculates based on how well your prediction is. Calculation of the mAP is based on Precision, Recall, IoU (Intersection Over Union).**IoU:**

IoU says how good your prediction bounding boxes are.

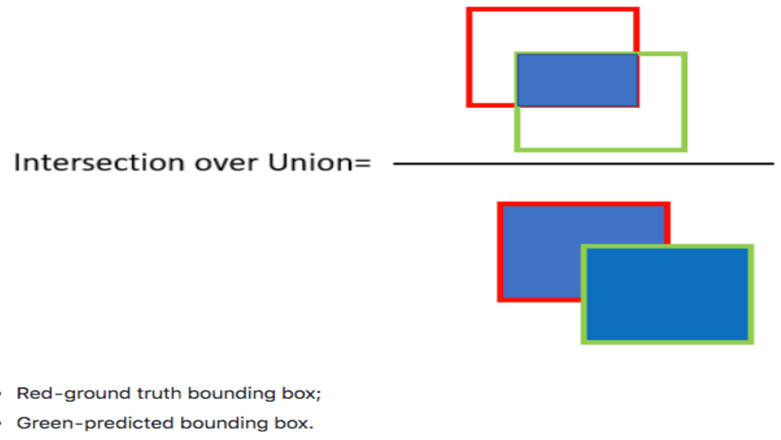

Intersection over Union=
• Red-ground truth bounding box;
• Green-predicted bounding box.

Figure 4.1.4.1: IoU (Intersection Over Union) Example [16]

Here Red boxes are actual label created during the labeling process and green box are the predicted boxes so IoU says how well our prediction fit to the actual image label.

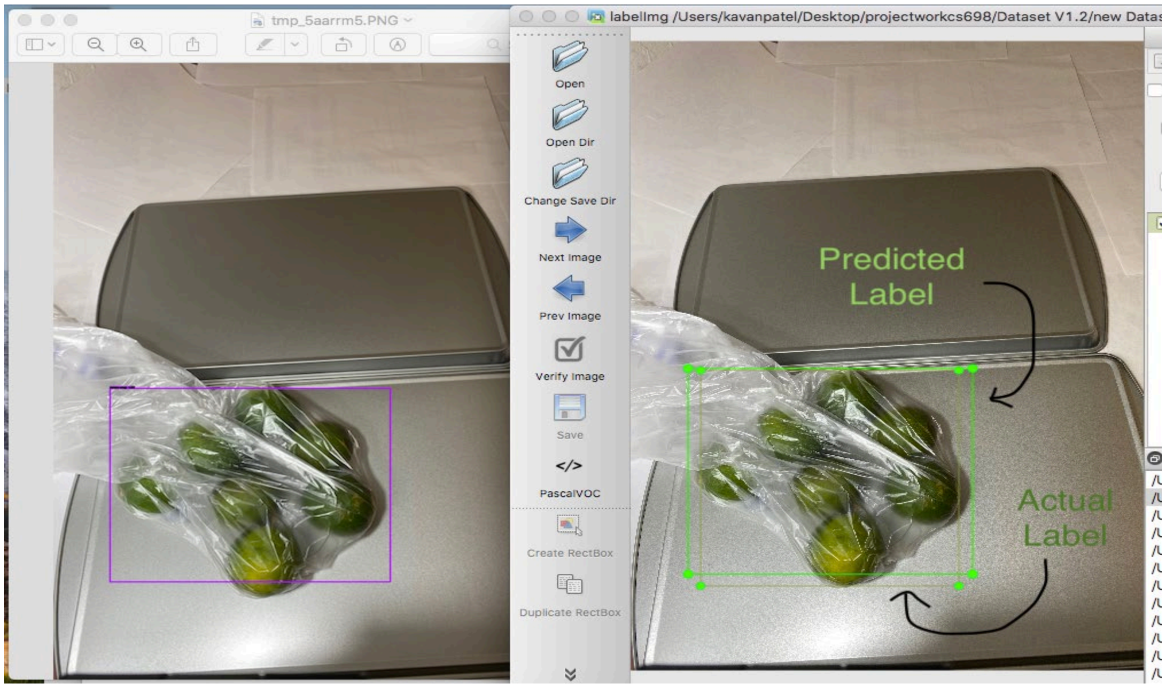


Figure 4.1.4.2 IoU (Intersection Over Union) in Real

Here, in figure 4.1.4.2 the dark green(Actual Label) marked label boxes are actual labels made during the dataset labeling process. And Light Green (Predicted Label) marked label boxes are predicted labels by our custom trained label.

Precision calculation:

→ if IoU is >= 0.5, it classifies as True Positive (TP) [16].

→ if IoU is <0.5, it classifies as False Positive (FP) [16].

→ When the label is present in the image and the model cannot be able to predict then it classifies as False Negative (FN) [16].

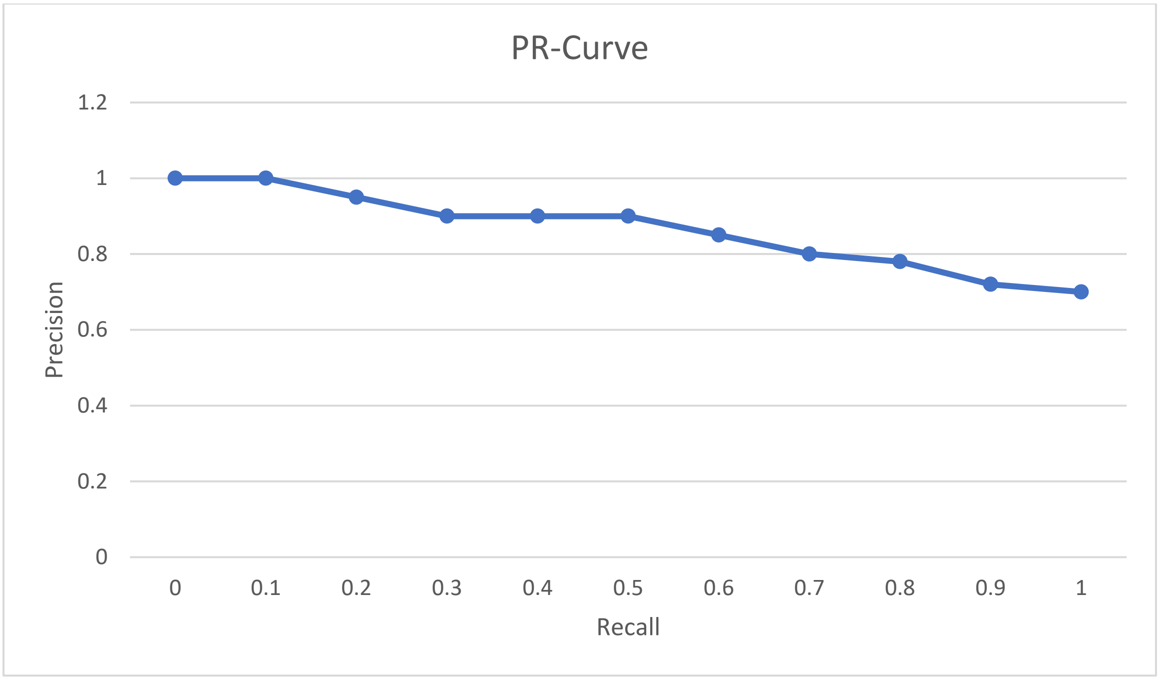
Precision P= TP / FP + TP

Recall R= TP / FP + FN

mAP= $1/N \sum_{1}^{N} APi$ where N is total number of images.

AP (Average Precision): It is an average value of precision at 11 points on PR-curve.

PR-Curve is made by Precision and Recall values. Where AP is calculated by average of maximum precision value at 11 recall point ranging from (0.0, 0.1, 0.2,…,1.0)



Graph 4.1.4.1 PR-Curve (Example)

To calculate AP (Average Precision) of above graph = 1/11 [1+1+0.95+0.9+0.9+0.9+0.85+0.8+0.78+0.72+0.7] = 0.86 (86%)

In mAP calculation the threshold value for IoU is by default 0.5, It can be changed upon a model.

4.1.5. Check mAP of generated weight files during training:

Note: mAP calculation discusses on next section

the mAP is the most important part to check any model's accuracy. During Yolov4 custom object detection model training, it saves our model every 1000 iteration on our defined path (usually in our google drive).

The ideal time for the model training is 6000 steps for better accuracy as per the documentation of yolov4 [15]. We can stop training at 2000 steps it will give fair results. Here we start training our model on Colab and let them run overnight around 12 hrs. and we go done until 2000 steps. After testing our model, we thought it needs to get 6000 steps to get more accuracy. So we kick start our training from the last saved .weight file and do this process two to three more times because colab provides free GPU for a limited time. After our training until 6000 steps, we calculated mAP for each of the saved models, and results from our mAP calculation are perfectly aligned with the YOLOv4 official document and we got proper accuracy on actual detection which we needed so we stopped our training at 6000 steps and move forward to next model conversion and deployment process.

Iteration	1000	2000	3000	4000	5000	6000	Best weight (6000 +)
mAP	0.7782 (77.82%)	0.9736 (97.36%)	0.9914 (99.14%)	0.9918 (99.18%)	0.9939 (99.39%)	0.9924 (99.24%)	0.9952 (99.52%)

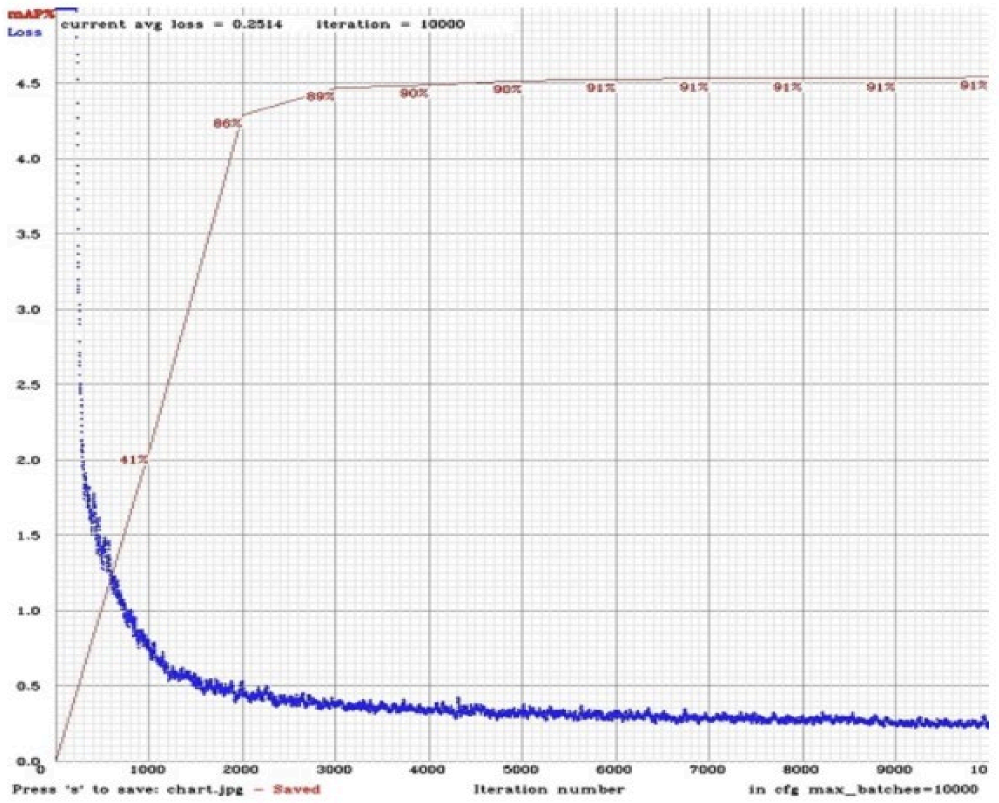


Figure 4.1.3 mAP vs Loss over iteration [15].

Here above table shows mAP calculation by our generated model and figure 4.1.3 shows mAP vs. Loss over iteration provided by official documentation if we match our table number with the graph it completely aligns with the graph for every 1000 iteration.

Here is the sample mAP Calculation from actual result at 6000+ iteration:

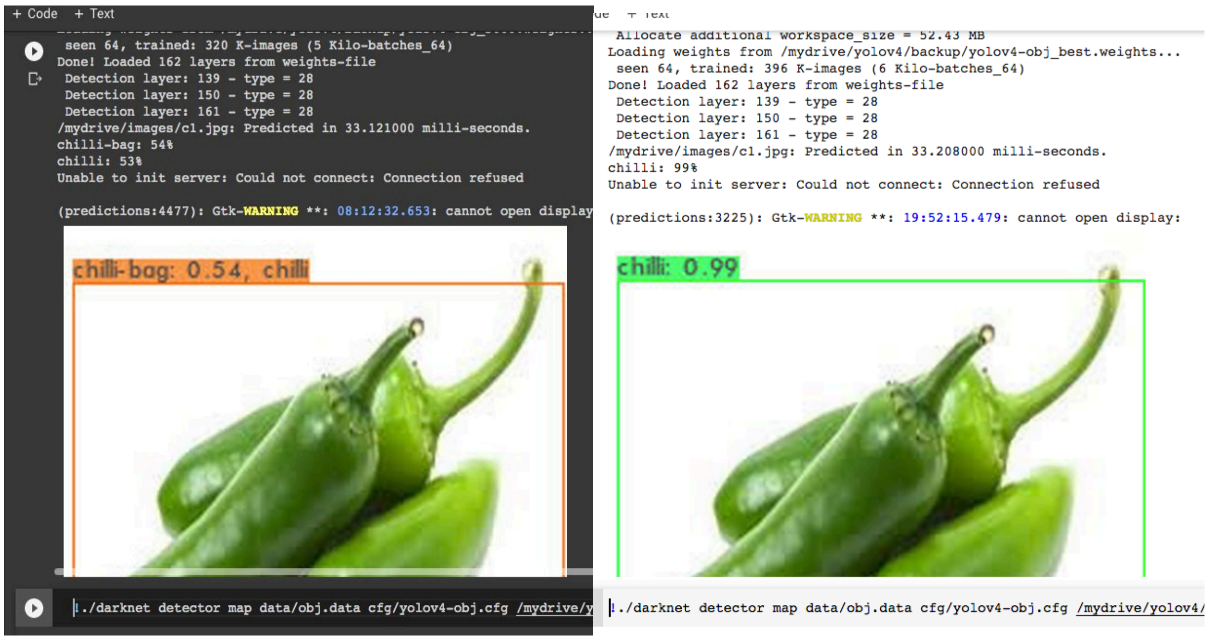
IoU threshold = 50 %, used Area-Under-Curve for each unique Recall

mean average precision (mAP @ 0.50) = 0.995275, pr 99.53 %

Total Detection Time: 109 Seconds

4.1.6. Use generated weight files to do actual detection with custom model:

Weight files are the model file which consists of weights for the detection. During the training, we got a weight file for every 1000 iteration and we test those files on images to see the results.



```
 seen 64, trained: 320 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/images/c1.jpg: Predicted in 33.121000 milli-seconds.
chilli-bag: 54%
chilli: 53%
Unable to init server: Could not connect: Connection refused

(predictions:4477): Gtk-WARNING **: 08:12:32.653: cannot open display
```

```
!./darknet detector map data/obj.data cfg/yolov4-obj.cfg /mydrive/y
```

```
 Allocate additional workspace_size = 52.43 MB
Loading weights from /mydrive/yolov4/backup/yolov4-obj_best.weights...
 seen 64, trained: 396 K-images (6 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/images/c1.jpg: Predicted in 33.208000 milli-seconds.
chilli: 99%
Unable to init server: Could not connect: Connection refused

(predictions:3225): Gtk-WARNING **: 19:52:15.479: cannot open display:
```

```
!./darknet detector map data/obj.data cfg/yolov4-obj.cfg /mydrive/yolov4/
```

Figure 4.1.4 Earlier Model Detection vs Recent Model Detection

In Figure 4.1.4 it is a comparison of the model detection at 2000 iteration vs model detection at 6000 iteration and we observed a huge difference in detection in the early stages.

4.1.7. Model Conversion

Yolov4 Custom Object detection model trained entirely on Darknet environment. It can perform detection on the darknet environment during the coding process but to use our model for real prediction It needs to be deployed on GUI.

TensorFlow provides many different methods to deploy our model into actual production. There is plenty of TensorFlow API used to convert our model as a TensorFlow model like TFJS, TFlite.

TFJS: It is the TensorFlow version of the deep learning model which is very lightweight and can be served through the web.

TFlite: It is the TensorFlow version of the deep learning model which is lightweight and used for mobile and IoT(Internet Of Things) Devices.

TensorFlow: It is a general-purpose TensorFlow version of the deep learning model.

Here, we used TensorFlow-Yolov4-Tflite API to convert our darknet Yolov4 model to the TensorFlow model. This conversion generates our TensorFlow version of the model with .pb (protobuf file)

Protobuf file is developed by google which contains graph information and weights of the model. Directory artifact of the YOLOv4 model After Converted to TensorFlow:

```
Yolov4-416
|
------/assets
------/variables
        |
        ------ variables.data-00000-of-00001
        ------ variables.index
------saved_model.pb
```

4.1.8. Deploy on GUI (Graphical User Interface)

Deployment is a step to serve the model available to others. The deployment model is simple as it takes input and gives an output (in our case image with prediction).

According to Study in data science study, 87% of the data science project never got into production. Lake of deployment can be one of the reasons where the project is developed but it never actually out from the console.

Deployment can be done in various ways on various platforms. For example, Google provides google cloud, Amazon provides an amazon sage maker for cloud-based machine learning model deployment.

Here for our project, local deployment is enough for our need as it reduces the cost of the cloud, so it makes no extra cost project.

Here, we used the Tkinter library of python, which is Great for GUI development.

We try to keep it simple, so there is a load weight button to load our model weight (model weights need to be load only once). Then load the image button to load our image and the Classify image button to predict it.

On press of the classify, it can request the TensorFlow version of the YOLOv4 model and get the prediction.

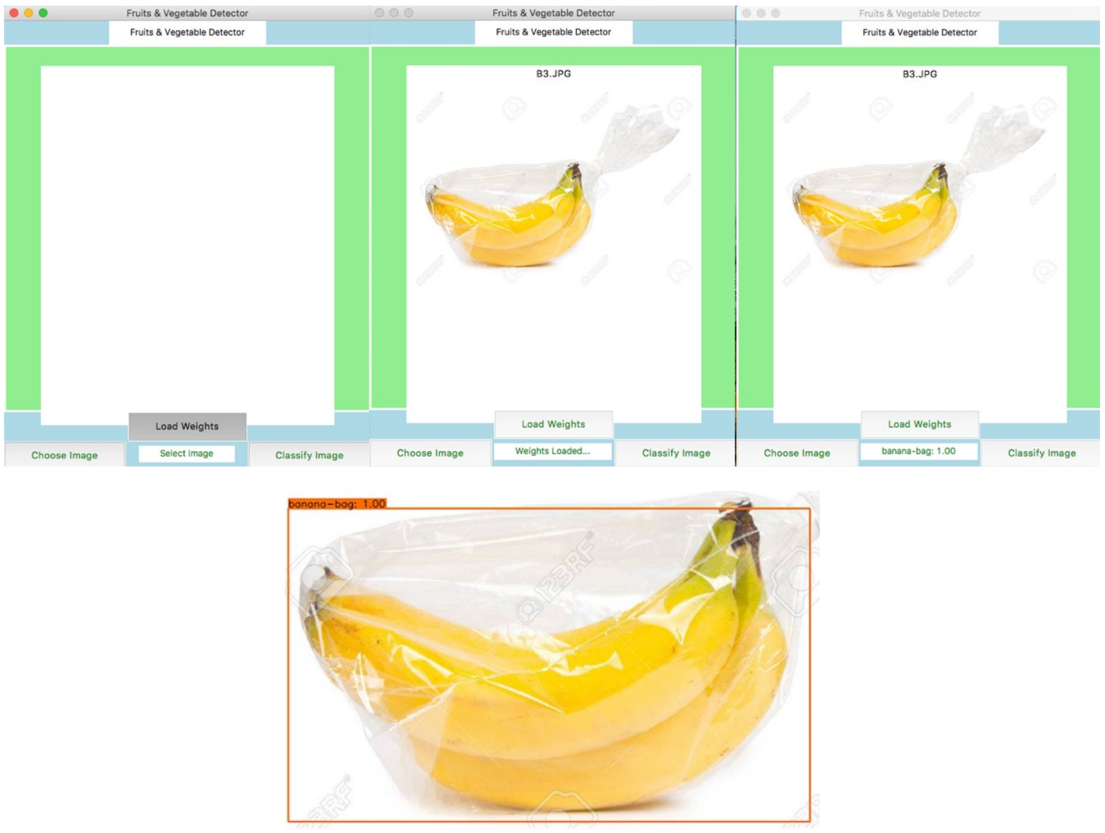


Figure 4.7 GUI Of Fruits & Vegetable Detection POS

5. Results and Discussions

After, all of the work, the results we got are amazing and up to the mark. Here, our trained model speed of detection excellent on GPU and CPU as well.

GPU Speed: 0.0200 to 0.0300 Second.

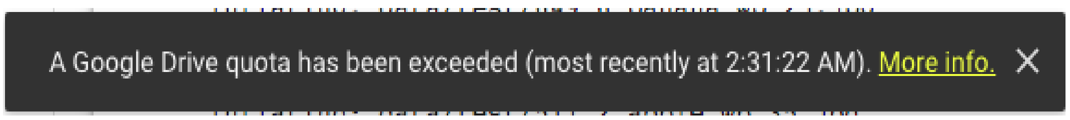
CPU Speed: Under 1 Second.

If we talk about performance, the initial models are struggled to give an accurate performance, it's around 40 to 50% accurate. After more training, the model improvement is amazing with almost 80% to 100% accurate.

5.1. Lesson Learned

- To training any model, the dataset collection is the most important part. For any model training, we need to plan for it. After our 2nd Iteration, we realized the data set we created is not enough to get a good model.
- During the custom data set creation, we need to keep the image size within a limit. Sometimes it's good to use an image compression tool like JPEG mini.

  JPEGmini: it's a great tool for image compression without losing the quality of the image. Here, we are performing the training part on the cloud. And store our dataset on google drive. Because of the big dataset size ~15GB after 1 run of program it gives download quota Finnish error from google drive. We cannot further transfer our data from google drive to Colab's Virtual machine. In this case, we need to wait up to 2 days to get reset the download quota limit.



A Google Drive quota has been exceeded (most recently at 2:31:22 AM). More info.

Figure 5.1 Drive Quota limit exceeded

  Solution: To get rid of the drive quota limit we used the JPEG mini tool and reduce our dataset size from 15Gb to 5Gb and we can resume our training.

- Check model accuracy some thousand iterations. If it is not aligning with the official documentation, there is some problem with the training.

5.2. Benefits of Adding ML on Current System

Hypothesis:

→ Let's assumed, for single fruit or vegetable item, the customer can spend an avg of 20-30 sec (It can be more for customers new to self-checkout).

→On avg 400 people visits supermarket every day among them 300 uses self-checkout

→Avg customer buys 3-5 fresh food items (it could be more)

If we math all of this Hypothesis it will be approx. 600 minutes

Proposed System.

→Our machine learning model can classify an image in a second. Assumed for a single item it can take up to 3-5 sec for all processes so it will be approx. 100 minutes.

By the above classification, we can say the new system can take significantly more customer than the current system

6. Conclusion and Future Work

6.1. Conclusion

By model analysis and accuracy performance, we can say that the generated model is giving enough results, for our fruits & vegetable detection POS purpose. mAP of the model is 0.94 which also says about model accuracy.

To load the weight in the application, it takes up to 30 sec on very less powerful CPU, which can be improved with the good CPU or GPU

During actual testing, for some of the images, we still got false prediction or no prediction in the case of the multiple objects. In the self-checkout station, there is a need to be only one item placed on the counter, so we can ignore the problems with the multiple objects, where the problem of the false prediction can be solved by post processing output.

6.2. Future Work

Here, we trained our model for only 14 different classes, upon the success of our model it can be trained mode class as per need by repeating the same process of custom model training.

Prediction from the live video is always an exciting part of any object detection model. Our model can do that but, it requires solid hardware and few ticks.

Sometimes as 5 to 10 % of times, it predicts as a false class. It can be eliminated by feeding more than one image in a system, during image feeding from the live video (e.g., we can capture five different images during image acquisition on the interval of 0.5 sec) make five different predictions and use the max function to choose output. That can improve real-time model accuracy more.

Developing a good GUI for better results is still part of future work.

References

1) AI Involvement in Supermarket: Food west problem
Blog (https://www.futurithmic.com/2019/06/10/forget-self-checkout-supermarkets-future-will-rely-on-ai/)
2) Cheating By changing Fruits or Vegetable name at self-checkout
Blog (https://www.dailymail.co.uk/news/article-5774293/Brits-using-self-checkout-machines-steal-expensive-fruit-veg-saying-theyre-buying-carrots.html)
3) Cheating By changing Barcodes at self-checkout
Blog (https://www.theatlantic.com/magazine/archive/2018/03/stealing-from-self-checkout/550940/)
4) YOLO: Real Time Object detection:
Blog (https://pjreddie.com/darknet/yolo/)
5) YOLO Versions & History:
Data Science (https://towardsdatascience.com/yolo-v4-or-yolo-v5-or-pp-yolo-dad8e40f7109)
6) Bochkovskiy, Alexey & Wang, Chien-Yao & Liao, Hong-yuan. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection.
Research Gate (https://www.researchgate.net/publication/340883401_YOLOv4_Optimal_Speed_and_Accuracy_of_Object_Detection)
7) Fruits and Vegetable Classification From Live Video
Danev, L. (2017). Fruit and vegetable classification from live video. Semantic Scholar (https://www.semanticscholar.org/paper/Fruit-and-vegetable-classification-from-live-video-Danev/e3c565f3b6b90f91017a844d0c664e85f6be52ca?p2df)
8) Tunnel-type digital imaging-based system for use in automated self-checkout and cashier-assisted checkout operations in retail store environments
Patents (https://patents.google.com/patent/US20090134221A1/en)
9) Tiliter Vision: AI Self-checkout
Blog (https://www.tiliter.com/blog/blog-post-title-two-kd2tg)
10) Simplifying Grocery Checkout with deep Learning:

CS230 Deep Learning (http://cs230.stanford.edu/projects_fall_2019/reports/26257432.pdf)

11) Rudnik, Katarzyna & Michalski, Pawel. (2019). A Vision-Based Method Utilizing Deep Convolutional Neural Networks for Fruit Variety Classification in Uncertainty Conditions of Retail Sales. Applied Sciences. 9. 3971. 10.3390/app9193971. Applied Sciences (https://www.researchgate.net/publication/335983502_A_Vision-Based_Method_Utilizing_Deep_Convolutional_Neural_Networks_for_Fruit_Variety_Classification_in_Uncertainty_Conditions_of_Retail_Sales)
12) CloDSA Dataset Augmentation: Casado-García, Á., Domínguez, C., García-Domínguez, M. *et al.* CLoDSA: a tool for augmentation in classification, localization, detection, semantic segmentation and instance segmentation tasks. *BMC Bioinformatics* **20,** 323 (2019).

    BMC Bioinformatics (https://doi.org/10.1186/s12859-019-2931-1)
13) Xml-to-Txt Conversion for YOLO object detection dataset.: GitHub (https://github.com/Isabek/XmlToTxt)
14) Darknet: an Opensource Neural Network Written in C Blog (https://pjreddie.com/darknet/)
15) YOLOv4_Darknet: YOLOv4 to train and evaluate custom model GitHub Directory **https://github.com/AlexeyAB/darknet**
16) Calculating mAP **Calculating mAP (https://medium.com/analytics-vidhya/understanding-the-map-mean-average-precision-evaluation-metric-for-object-detection-432f5cca53b7)**
17) Tensorflow-yolov4-tflite(API): Yolo v4 weights to .pb TensorFlow serving converting **GitHub (https://github.com/hunglc007/tensorflow-yolov4-tflite)**

Appendix

Tools & Libraries Used:

- Python 3.7
- Tkinter: For making GUI Using python
- OpenCV: an open-source library for Computer Vision
- Tensorflow-Yolov4-Tflite: Converting YOLOv4 Model to TensorFlow Lite Model
- NumPy: Math Operation (Operation with tensor)
- TensorFlow: an open-source machine learning library
- LabelImg: A tool to Label Customize dataset
- Colab: A Cloud IDE for developing code on Cloud and use powerful GPU
- Google Drive: To store custom dataset for the training
- JPEG mini: Tool to compress the image size without loosing quality
- PIL(Pillow): A Python free Open-source image Library