IOTIOT

Co-Op Internship Report

Grocery Product Recognition

SEPTEMBER - DECEMBER 2020

Gautam Jagdhish
CSE, IIT Dharwad

Table of contents

Abstract

The objective of this internship is to develop a grocery product recognition system which is able to detect the grocery products in an image. I have used an object detection model, YOLO V4 and MVTec D2S Grocery Database to train the Neural network and was able to achieve a mAP score of 61 percent on the test dataset.

Introduction

A grocery product recognition system can have multiple uses including Real time Shelf Management and Automatic Checkout System. Real time Shelf Management is useful to observe which shelves are empty and where restocking is required. Automatic Checkout(ACO) Technology automatically generates shopping lists from images of products for purchase at brick-and-mortar stores, vastly improving the efficiency of traditional shopping experiences. Using similar technologies, Amazon operates Amazon Go stores which are cashier-less. The main challenge of developing computer vision models for retail use cases comes from the difficulty of collecting sufficient training data to reflect realistic checkout scenarios.

Training Setup

The Computational power needed to train machine learning and deep learning models on large datasets, has always been a huge hindrance for machine learning enthusiasts. But, with the introduction of cloud hosted Jupyter Notebooks by major tech companies, anyone with just an Internet connection and Machine Learning knowledge is able to train and test ML and DL models for free such as,

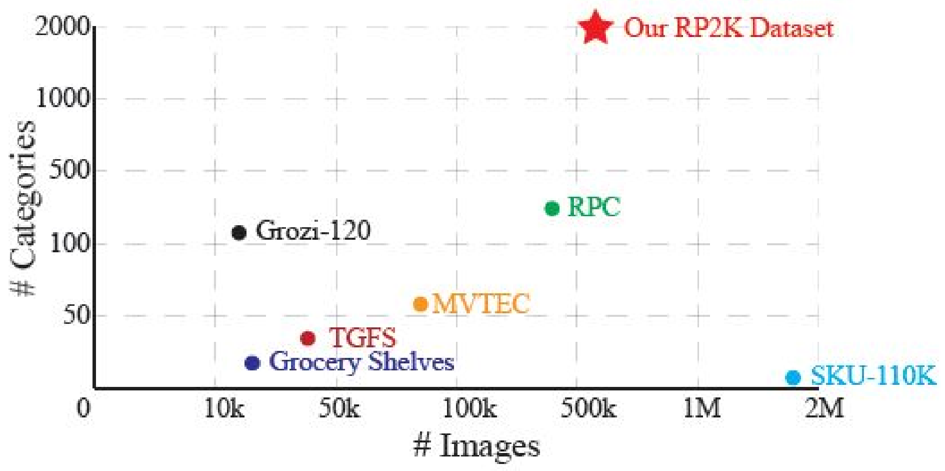
1. Google Colab
2. Kaggle Kernel
3. Jupyter Notebook on GCP
4. Amazon SageMaker
5. Azure Notebooks

All the above services have some kind of limitations in their free tier.

Also, since I am at home with limited internet connection, downloading huge datasets is also a concern. So, I need a service which also offers a good amount of storage to download, process the dataset and save the trained model and weights online. Luckily, my college provides every student with a GSuite account with unlimited storage in Google Drive. Considering the performance and storage limitations of the above free services, I chose Google Colab as the training platform. So, in Colab I mounted my Google drive to download, unzip and process the dataset then trained and tested the model for free!

Choosing the Dataset

Most of the large-scale datasets such as Google AI Open Images, ImageNet, COCO, Youtube-8M focus on everyday photography or urban street scenes, which makes them of limited use for many industrial applications. Furthermore, the amount and diversity of labelled training data is usually much lower in industrial settings. To train a visual warehouse system, for instance, the user typically only has a handful of images of each product in a fixed setting. Nevertheless, at runtime, the products need to be robustly detected in very diverse settings. Only a few datasets focus on industry-relevant challenges in the context of warehouses. The Freiburg Groceries Dataset, SOIL-47, and the Supermarket Produce Dataset contain images of supermarket products, but only provide class annotations on image level. The MVTec D2S Grocery dataset, RP2K, Grocery Products Dataset and GroZi-120 include bounding box annotations that can be used for object detection. However, in Grocery Products Dataset and GroZi-120 not all object instances in the images are labeled separately.



So, the datasets on grocery products with bounding box annotations for each instance of an object are MVTec D2S Grocery dataset and RP2K. MVTec's grocery dataset has 7980 images with boundary box information for 60 products whereas, RP2K has 500,000 images on 2000 products. However, there are some issues with the RP2K dataset website due to which the dataset was not accessible. Also, training half a million images on limited resources is also not viable. So, the MVTec D2S Grocery dataset is **chosen**.

MVTec D2S Supermarket dataset

Out of the 7980 images which are labelled in the MVTec dataset, 4380 images only contain objects of a single class on a homogeneous background are used for training, while the remaining 3600 images are much more complex and diverse. So, I split the 3600 diverse images into a 50-50 split for validation and testing. Now, dataset is split into :

1. Training - 4380 images
2. Validation -1800 images
3. Test - 1800 images

Training dataset :

- Has same background
- No clutter
- 6900 objects in 4380 images
- Average number of objects per image is 1.58

Validation and testing datasets :

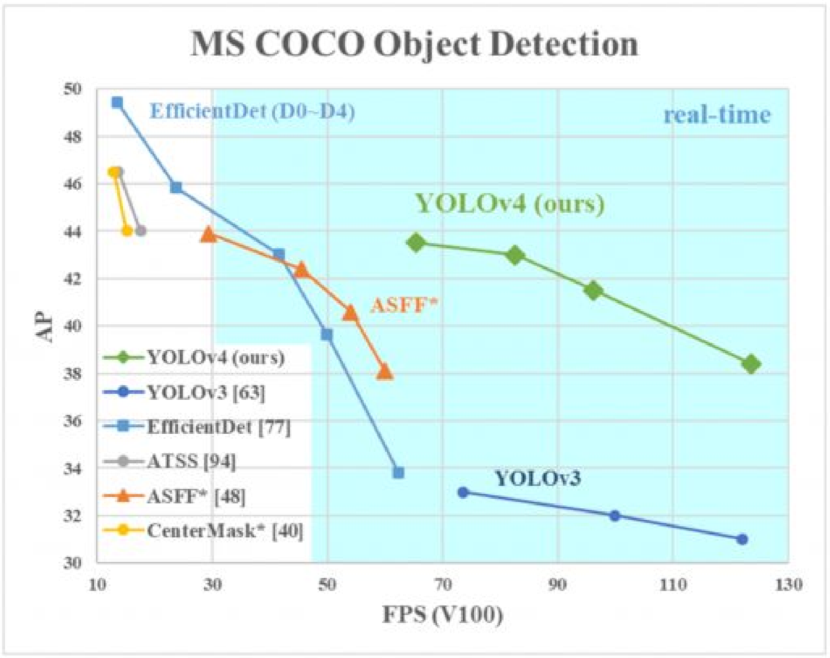
- Has variation in background
- Has clutter
- 15654 objects in 3600 images
- Average number of objects per image is 4.35
- More number of images with occlusion

Choosing the Model

An object detection algorithm is required which also predicts bounding boxes on each instance of an object in an image. There are several object detection algorithms with different capabilities. These algorithms are mostly split into two groups according to how they perform their tasks. The first group is composed of algorithms based on classification and work in two stages. First, they select the interesting parts of the image, and then they classify objects within those regions using Convolutional Neural Networks (CNN) . This group, which includes solutions such as R-CNN, is usually too slow to be applied in real-time situations. The algorithms in the second group are based on regression - they scan the whole image and make predictions to localize, identify and classify objects within the image. Algorithms in this group, such as You Only Look Once (YOLO), are faster and can be used for real-time object detection. The top algorithms in this field of computer vision are,

- Fast R-CNN
- Faster R-CNN
- Histogram of Oriented Gradients (HOG)
- Region-based Convolutional Neural Networks (R-CNN)
- Region-based Fully Convolutional Network (R-FCN)
- Single Shot Detector (SSD)
- Spatial Pyramid Pooling (SPP-net)
- YOLO (You Only Look Once)

Considering the tradeoffs between accuracy and inference time of various models, YOLO V4 is in the sweet spot with the state of the art results : 43.5% AP(65.7% AP_{50}) at a real time speed of 65 frames per second on Tesla V100.



Why YOLO ?

You Only Look Once (YOLO) is a network that uses Deep Learning (DL) algorithms for object detection. YOLO performs object detection by classifying certain objects within the image and determining where they are located on it. Previous object detection methods like Region-Convolutional Neural Networks (R-CNN), including other variations of it like fast R-CNN, performed object detection tasks in a pipeline of multi-step series. R-CNN focuses on a specific region within the image and trains each individual component separately. This process requires the R-CNN to classify 2000 regions per image, which makes it very time-consuming (47 seconds per individual test image). Thus it cannot be implemented in real-time. Additionally, R-CNN uses a fixed selective algorithm, which means no learning process occurs during this stage so the network might generate an inferior region proposal. This makes object detection networks such as R-CNN harder to optimize and slower compared to YOLO.
A YOLO network consists of three main parts. First, the algorithm, also known as the predictions vector. Second, the network. Third, the loss function.

How does YOLO work ?

The YOLO Algorithm

Once you input an image into a YOLO algorithm, it splits the images into an SxS grid that it uses to predict whether the specific bounding box contains the object (or parts of it) and then uses this information to predict a class for the object.

How the algorithm builds and specifies each bounding box ?

The YOLO algorithm uses four components and additional value to predict an output :

1. The center of a bounding box (bx by)
2. Width (bw)
3. Height (bh)
4. The Class of the object (c)

The final predicted value is confidence (pc). It represents the probability of the existence of an object within the bounding box. The (x,y) coordinates represent the center of the bounding box. Typically, most of the bounding boxes will not contain an object, so pc prediction is used. Using a process called non-max suppression, unnecessary boxes with low probability and those who share big areas with other boxes are removed

The Network

A YOLO network is structured like a regular CNN, it contains convolutional and max-pooling layers and then two fully connected CNN layers.

The Loss Function

Only one of the bounding boxes should be responsible for the object within the image since the YOLO algorithm predicts multiple bounding boxes for each grid cell. To achieve this, the loss function computes the loss for each true positive. To make the loss function more efficient, the bounding box with the highest Intersection over Union (IoU) with the ground truth is selected. This method improves predictions by making specialized bounding boxes which improves the predictions for some aspect ratios and sizes.

A Brief History of YOLOs

The original YOLO (You Only Look Once) was written by Joseph Redmon (now retired from Computer Vision) in a custom framework called Darknet. Darknet is a very flexible research framework written in low level languages and has produced a series of the best real time object detectors in computer vision: YOLO, YOLOv2, YOLOv3, and now, YOLOv4.

The Original YOLO

YOLO was the first object detection network to combine the problem of drawing bounding boxes and identifying class labels in one end-to-end differentiable network.

YOLOv2

YOLOv2 made a number of iterative improvements on top of YOLO including BatchNorm, higher resolution, and anchor boxes.

YOLOv3

YOLOv3 built upon previous models by adding an objectness score to bounding box prediction, added connections to the backbone network layers, and made predictions at three separate levels of granularity to improve performance on smaller objects.

Now, onto YOLOv4,

YOLO V4

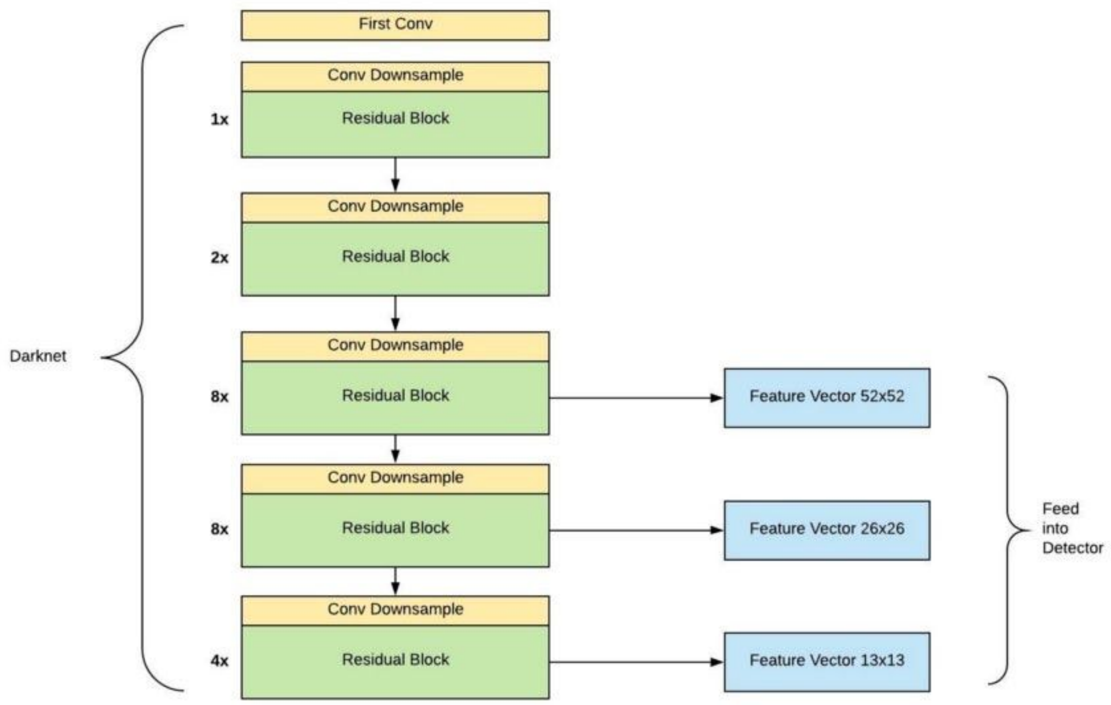
YOLO V4 is an important improvement of YOLO V3, the implementation of a new architecture in the *Backbone* and the modifications in the *Neck* have improved the mAP(mean Average Precision) by 10% and the number of FPS(Frame per Second) by 12%. YOLOv4 makes real time detection a priority and conducts training on a single GPU. The authors' intention is for vision engineers and developers to easily use their YOLOv4 framework in custom domains.
A modern detector is usually composed of three parts :

1. **Backbone - Feature Formation**
   Backbone is a deep neural network composed mainly of convolution layers. The main objective of the backbone is to extract the essential features, the selection of the backbone is a key step it will improve the performance of object detection. Often pre-trained neural networks are used to train the backbone.
   In the case of YOLOv4, the authors considered the following backbones for the YOLOv4 object detector :
   - CSPResNext50
   - CSPDarknet53
   - EfficientNet-B3

   Based on their intuition and experimental result, the final YOLOv4 network implements **CSPDarknet53** for the backbone network
2. **Neck - Feature Aggregation**
   The next step in object detection is to mix and combine the features formed in the ConvNet backbone to prepare for the detection step. YOLOv4 considers a few options for the neck including:
   - FPN
   - PAN
   - NAS-FPN
   - BiFPN
   - ASFF
   - SFAM

   The components of the neck typically flow up and down among layers and connect only the few layers at the end of the convolutional network. YOLOv4 chooses **PANet** for the feature aggregation of the network.

3. **Head - The Detection step**
   Detection happens in the head using the extracted features from the backbone as input. It is usually categorized into two kinds, i.e., one-stage object detector and two-stage object detector. The most representative two-stage object detector is the R-CNN series whereas the most representative two-stage detectors are YOLO series, SSD and RetinaNet.
   YOLOv3 revisit :
   YOLOv3's object detection system can be divided into two major components: Feature Extractor and Detector, both are multi-scale. When a new image comes in, it goes through the feature extractor first so that we can obtain feature embeddings at 3 different scales. Then, these features are fed into 3 branches of the detector to get bounding boxes and class information. If the input size is 416x416, the three scale vectors would be 52x52, 26x26, and 13x13.
   YOLOv4 deploys the same **YOLO** head as YOLOv3 for detection with the anchor based detection steps, and three levels of detection granularity.



Training and testing

Transfer Learning by freezing the convolutional base

The main idea here is to keep the convolutional base of a pre trained model in its original form and then use its outputs to feed the classifier. The pre-trained model is used as a fixed feature extraction mechanism. This method is ideal when we are short on computational power or the dataset is small and/or a pre-trained model solves a problem very similar to the one you want to solve.

Getting Familiar with darknet

To get familiar with the usage of darknet and transfer learning on YOLOv4, I initially chose a smaller dataset. So, I downloaded 300 images each from 5 classes for a total of 1500 from the Google Open Images Dataset, converted them to the YOLO input format and modified the YOLOv4 configuration files accordingly. The pre-trained convolutional weights of the COCO dataset are selected. Since the dataset, config file and the weight file are ready for training, this model was trained for 10000 iterations.

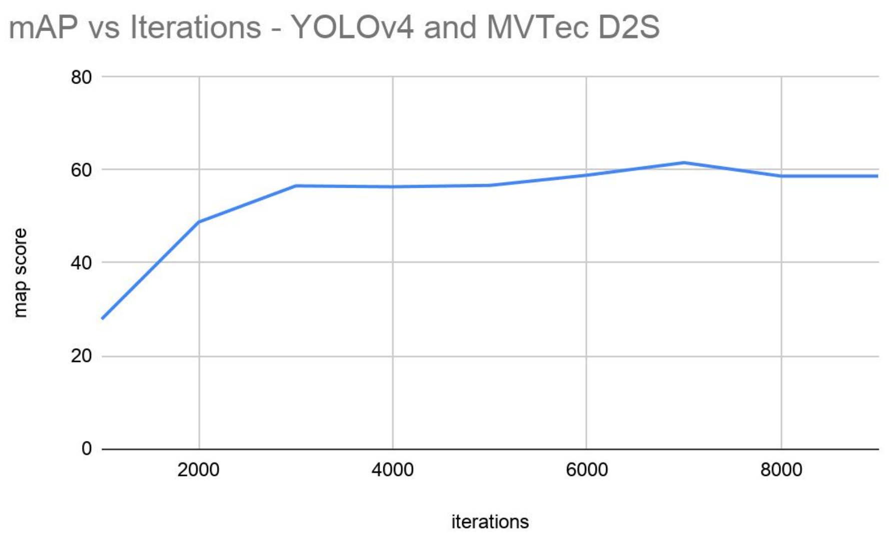
Choosing the best weights file

In darknet by default, the weights are updated after every 100 iterations and get saved after every 1000 iterations. The weights file that gets saved after the most number of iterations doesn't necessarily mean that it's the most accurate. This is due to overfitting of data. So, by comparing the mAP scores of these weights files against the test dataset, the best weights file is selected. For this dataset, the weights file which was saved at 3000 iterations had the best mAP score.

Training on MVTec D2S

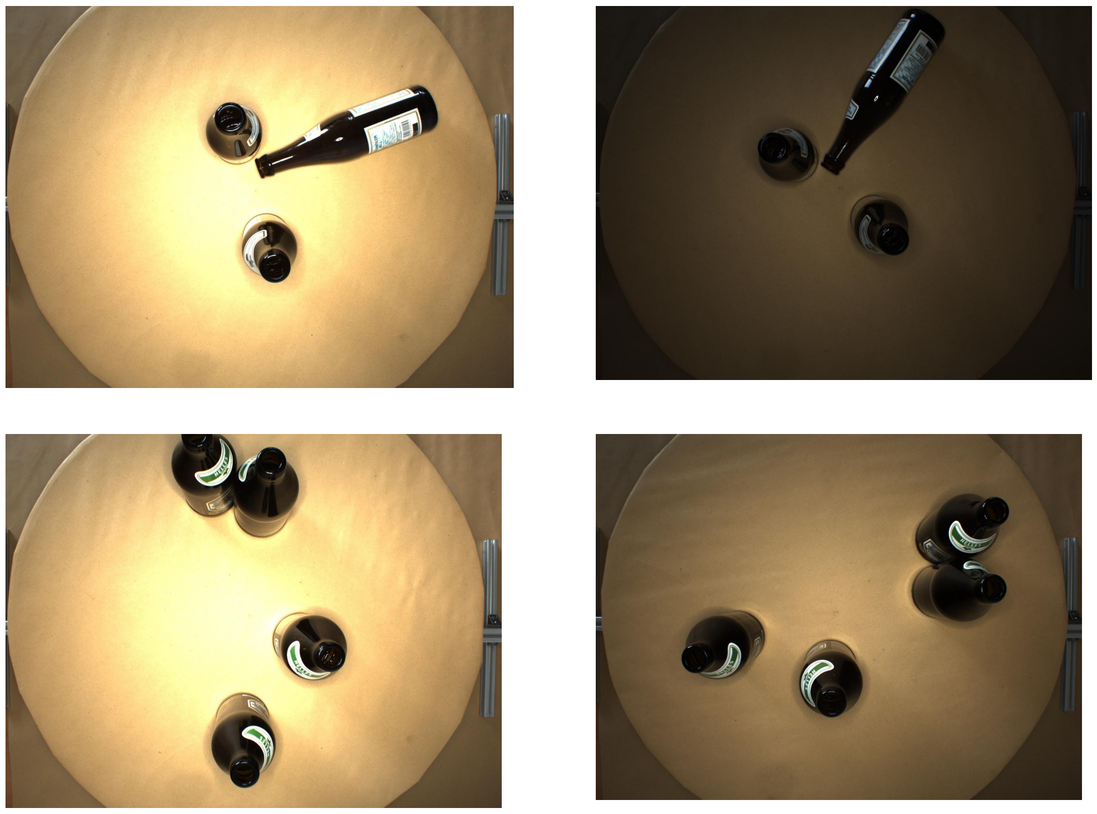
Now, after gaining some experience on running a custom object detector on darknet and YOLOv4, I started to train on the MVTec D2S dataset. So, the MVTec D2S dataset is converted to the YOLO format, config file is modified accordingly and the same initial convolutional weights file is used. Now, all the necessary files are ready. Training for 10000 iterations on this dataset took approximately 10 hours.

Then, the saved weights files are tested against the test dataset and the mAP values are recorded. The mAP scores gradually increased till 7000 iterations and started to decrease afterwards due to overfitting on the training data. So, the weights file which was saved after 7000 iterations is chosen as the best weights file.



Few examples

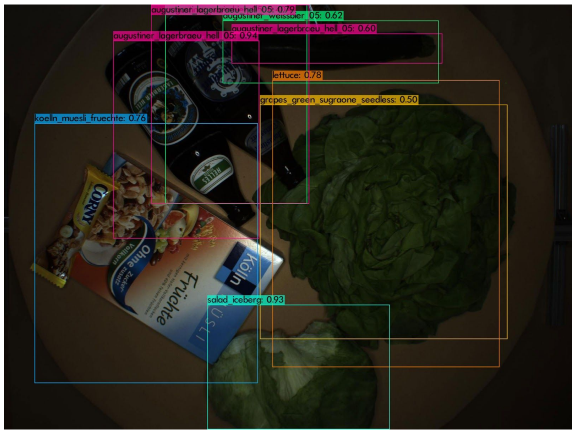
Training dataset only has a variations of lighting and positions of the product

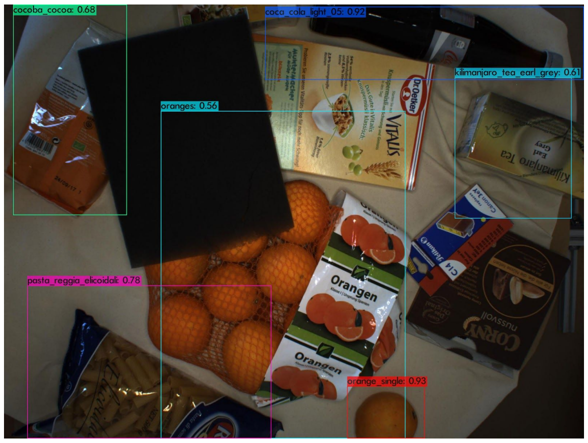


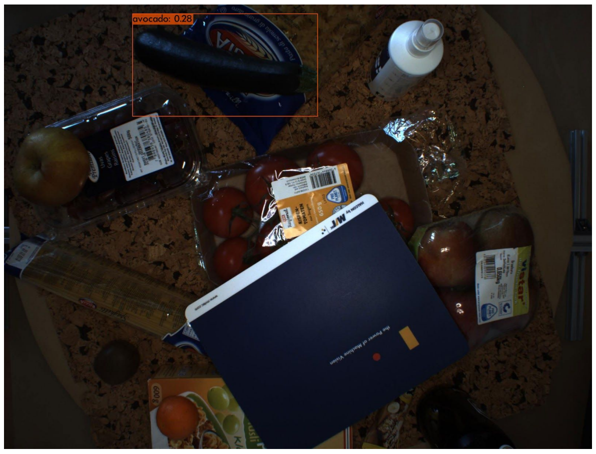
Test dataset

Different background, diverse images with clutter and occlusion.

Even though YOLOv4 was able to detect most of the products correctly in the first 2 test images shown below, it detected only one object on the third image


augustiner_lagerbraeu_hell_05: 0.79
augustiner_weissbier_05: 0.62
augustiner_lagerbraeu_hell_05: 0.60
augustiner_lagerbraeu_hell_05: 0.94
lettuce: 0.78
grapes_green_sugraone_seedless: 0.50
koelln_muesli_fruechte: 0.76
salad_iceberg: 0.93





Conclusion

In conclusion, this internship was a useful experience. I gained new knowledge and skills in the Computer Vision domain. Over the course of this internship I learned about different Object Detection models and was able to implement one. One of the main challenges was to find a dataset for this project and realise the scarcity of publicly available datasets for Grocery Products. I have also learned about the different comparison metrics available for object detection models. Two main life skills I learned are the importance of time-management skills and self-motivation.

References

1. MVTec D2S paper : https://arxiv.org/pdf/1804.08292
2. RP2K paper : https://arxiv.org/pdf/2006.12634
3. YOLO paper : https://arxiv.org/pdf/1506.02640
4. YOLOv3 paper : https://arxiv.org/pdf/1804.02767
5. YOLOv4 paper : https://arxiv.org/pdf/2004.10934
6. Darknet Framework : https://github.com/AlexeyAB/darknet
7. Google Open Images Dataset V6 : https://storage.googleapis.com/openimages/web/index.html

Some Articles :

8. https://missinglink.ai/guides/computer-vision/yolo-deep-learning-dont-think-twice/
9. https://blog.roboflow.com/a-thorough-breakdown-of-yolov4/
10. https://towardsdatascience.com/dive-really-deep-into-yolo-v3-a-beginners-guide-9e3d2666280e
11. https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751