Computer Vision Based Batch-Billing System For Supermarket Products Using YOLO

Submitted By

Exam Roll: 29913

Reg No: 2016615027

Session: 2016-17

Exam Roll: 29957

Reg No: 2016315002

Session: 2016-17

DEPT. OF ELECTRICAL AND ELECTRONIC ENGINEERING
UNIVERSITY OF DHAKA

Abstract

We propose a computer vision-based billing system for the check-out of retail items on the cash counter. Supermarkets employ traditional barcode scanners to detect retail products. The disadvantage of this prevailing check-out system is that it cannot detect a batch of products simultaneously and is hence time-consuming. The system also relies on the speed of the cashier's workflow in handling the cash counter. In this regard, our proposed system aims to make batch-billing of retail items faster and easier, providing a quality shopping experience for the customers. We have implemented a GUI-based check-out application that works on a YOLO-based object detection architecture to bill the retail products. With YOLO architecture at its core, the system scans retail items using a webcam placed above for detection. For training our model, we have chosen 25 local retail products and 5 weight-dependent products of various categories and used both authentic and synthetic images of them as our dataset. The detection of weight-dependent products is a challenging task since conventional packaging uses QR codes which cannot be detected optimally from long distances. We have overcome this problem by integrating Hybrid ArUco Markers in their packaging instead of QR codes. Overall, our system has achieved a reasonable real-time retail product detection accuracy to be implemented on the billing system paving the way for a fully automated supermarket experience inside the grocery stores.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Supermarkets or departmental stores are an integral part of human life all across the world. A lot of people go there in order to shop for their daily necessities. With a lot of people comes the problem of long queuing times in the checkout corner. Most supermarkets implement traditional infrared barcode-based scanners for billing. One case study of a Brazilian supermarket shows that a customer has to wait for 1.9-1.6 minutes on average in the queue and their checkout process takes around 4.5 minutes of service time from an average cashier.[12] While increasing the number of cashiers can reduce queuing time, the service time is completely dependent on the cashier's speed. This project aims to speed up checkout time by replacing traditional barcode scanners with a computer vision (CV) based checkout solution based on deep learning.

1.1 Motivation

Supermarkets offer a huge array of different variety products. Most stores use traditional billing methods based on infrared barcode scanners. Usage of bar code scanners is heavily time-consuming due to the workflow of using the scanners. The steps a cashier has to perform in order to scan the products are as follows:

- Pick up a product from the checkout area

- Search around the packaging for the barcode
- Scan it with a barcode scanner, if for some reason the scanner isn't immediately scanning the code, then the cashier has to reorient the scanner or the product to aid the scanning.

Picking up and searching for bar codes on each product individually, take up a lot of time. The process could be a lot faster if a batch-billing method is implemented where a batch of products will be billed at once instead of billing them individually. With the rise of deep learning-based object detection algorithms, image processing-based batch-billing of supermarket goods is becoming more and more viable. Companies like Amazon are already implementing such technologies in shopping carts called Dash cart[13]. But adding a computer-based system to each shopping cart is a very expensive endeavour and is not suitable for implementation in developing countries. So in this project, we aim to make a more viable solution by proposing a computer vision-based billing system at checkout corners.

1.2 Objectives

We aim to build a CV-based billing system for supermarkets. It is worth mentioning that the system we propose is not a self-checkout system. A cashier is still required to bill the products, we only aim to replace traditional barcode scanning with a CV-based solution that enables batch-billing. The objectives we aim to achieve are as follows:

- Implementing a CV-based system that can correctly classify supermarket products in batches with a view to reducing checkout time.
- Ensuring lighting independent product detection
- Ensuring that products can be detected both from their front and back labels making the system orientation independent to a certain extent
- Ensuring that subtle variants of products can also be distinguished

- Implementing a system to detect weight dependent products from images
- Creating a dataset of locally available supermarket products
- Implementing the system in a GUI-based application for ease of use

CHAPTER 2

RELATED WORKS

2.1 ARC Vision-based auto retail

In this study [14], the authors implemented a motor-driven conveyor setup that carries one product at a time inside a wooden hood, which has a webcam placed over its roof. Inside the hood, a Laser LDR (Light Dependent Resistor) module sends signals to a microcontroller board if a product has arrived inside or not. Depending on these signals, the board controls the switching of the conveyor motor. On detecting the product, the microcontroller communicates with the Python environment on the computer to access the webcam to extract a frame and process the image with OpenCV. In the processed image, they used Keras for object identification. They developed a GUI system using Tkinter that facilitates adding the identified products to the shopping cart with their price. In the CNN architecture, they employed ReLUs(Rectified Linear Units) [15] as the activation function, max-pooling, weight initialization [16], dropout [17] and batch normalization. For their training samples, they used 31,000 images of 100 different local retail products, and it produced a training accuracy of 94.76%, validation accuracy of 95.24%, and overall accuracy of 91.7%.

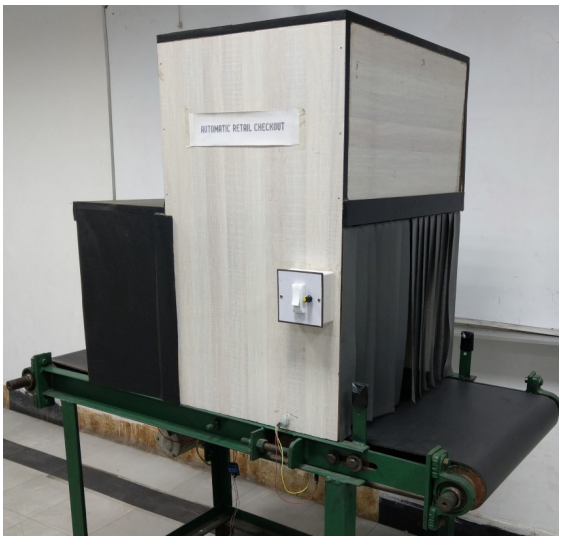


Figure 2.1: Structure of the hardware used in the paper

Limitations:

- Product detection one at a time consumes too much time and hence doesn't meet the goal of replacing the current manual retail check-out system.
- The limited size of the wooden hood restricts the size of the products.
- The system is limited to detecting the product in an ideal alignment condition.

2.2 A Computer Vision System Supporting Blind People

This study [18]proposes e-vision utilizing a computer vision application for going to the supermarket to buy food considering handicaps faced by blind people. Despite significant advances, assistive devices based on CV technology for vision-impaired persons are still limited to recognizing barriers and general objects without taking into account the context of the individual's actions. The functional requirements of an assistive device are considerably different in this scenario.

In order to provide a user-friendly experience, they examine the abstraction levels of information that must be communicated in the exhibited system while visiting

a supermarket. For example, If the user is looking at a trail, shelf, or product, or if they are at the supermarket's entrance/exit. The user would require various amounts of information at each abstraction level, on the trail information, the system should be able to indicate the specific trail you are on, for example, the drinks trail, added with the information of the level you are looking at i.e. the beer level and should be able to specify you about the brand of the beer you're holding- on the product level.

For the hardware part, this study uses three sensory devices:

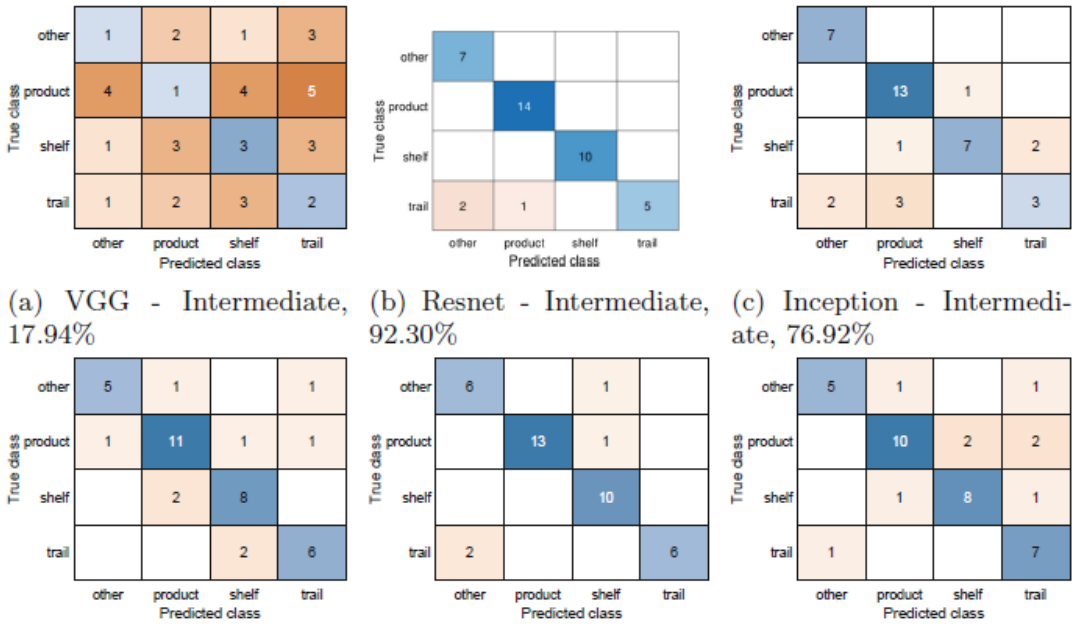
1. On the person's head or glasses, or on any body part on the same level of height, a camera is set.
2. To get images a mobile device application is used that the user carries, which gets the information through the designed methodology.
3. To communicate auditory information, a set of earphones is used, with pairs with the mobile application, giving feedback from the camera feed, which is processed through the mobile application, using text-to-speech technology.

The information extracted from the image feed is distinguished in 4 abstract levels, namely Product, Shelf, Trails, Other. The trails give the user the information of what section of products they are in front of. The shelf is information of the user being in front of a specific category of the above-mentioned products. And at the base level, the abstract Product communicates the information on the specific product (i.e. its specific name with its brand).

(a) Product (b) Shelf (c) Trail (d) Other

Figure 2.2: Sample images for different levels of abstract

For achieving this critical goal, this article makes use of the improvised Deep Neural Network(DNN) paired with Support Vector Machines (SVMs). Comparing three DNN architectures, VGG16 [19], ResNet [20] and Inception [21], pre-trained on ImageNet Datasets, the features from images were extracted. After categorizing the image on an abstract level, a mechanism is used to extract the accessible texts from the product packages, namely Optical CharacterRecognition (OCR) mechanism which can identify characters both in English and Greek.



(a) VGG - Intermediate, 17.94% (b) Resnet - Intermediate, 92.30% (c) Inception - Intermediate, 76.92%

(d) VGG - FC, 76.92% (e) Resnet - FC, 89.74% (f) Inception - FC, 76.92%

Figure 2.3: Representation of Confusion matrices for the quaternary classification task using VGG, Resnet, Inception

This article achieved an accuracy of 77.15% in communicating the information to the user.

Limitations:

- Inadequate dataset for on-site visits that limits the accuracy improvement
- "Other" cases are not defined properly (i.e. for shopping carts or employees)
- A visually assisting system build for ease of payment is not included

- Needs a system for differentiating packaged and non-packaged products

2.3 Toward Real-Time Grocery Detection for the Visually Impaired

The study [22] proposes ShelfScanner, an object detection system for visually impaired customers facilitating them to shop for groceries without asking for another human guidance. This system can detect products on the shelves in real-time from the shopping list of blind people by scanning a store area using only a mobile phone. It implements a translational motion model considering the approximate planar nature of the grocery shelves. Their training data consists of in-vitro images collected from the GroZi-120 database that provides 5.6 images on average per product. Their testing data consists of in-situ images captured from live video of the mobile camera. The system works by detecting a probable set of points from the camera's FOV(Field of vision) when these match any product of the shopping list and notifies the customer. ShelfScanner can also guide the customer for the whereabouts of any previously detected product by implementing a mosaic [23], which is continually being updated by adding new frames data with the running FOV. They did a cycle of the online evaluation for their system, which employs the Lucas-Kanade optical flow method(mosaicing) by OpenCV, SURF for interest point detection [24], NIMBLE [25] for estimation of the probability distribution over the shopping list classes, keypoint selection by comparing it with a threshold. Comparing with a strict threshold, out of 52 item classes, their system identified 17 easy items with zero false positives, eight moderate items with 1-100 false positives and hard items having more than 100 false positives.

Figure 2.4: Under-construction state of the mosaic represented by the green box in the FOV(Field of Vision) of the camera

(a) (b)

Figure 2.5: Training (top) and testing (bottom) examples of (a) easy retail items, which gives zero false positives for a strict threshold; (b) hard retail items, which fails to detect with reasonable false positives

Limitations:

- The training data are dependent on GroZi-120, which isn't always up to date with the current product appearances.

- The system operates at 2 FPS which is too slow for real-time application.
- As the system maintains a large mosaic for the slow FPS, the memory cost is high.
- Implementation of translational motion model affects object detection.
- The training images are obtained in ideal conditions but not in the real environment, for which the system couldn't detect hard retail items with reasonable false positives.

2.4 Fine-Grained Grocery Product Recognition by One-Shot Learning

This study [26] introduces retail product detection by combining feature matching with single-shot deep learning algorithm using a camera in the grocery stores. They implemented an unsupervised learning algorithm [27] for detecting the recurring features between the captured image of the product on the shelf and the sample image of the single-shot training algorithm. They produced a candidate ROI (Region of Interest) of the product on the shelf that contains only the distinguished features like the logo area and disregard the non-salient (redundant) attributes. They used the RANSAC algorithm [28] to align the feature map of the candidate ROI with the training sample to generate an attention map. The attention map magnifies the fine attributes of the product and feeds the data to VGG-16 (CNN classifier). They collected their dataset from GroZi-3.2 [29], GroZi-120 [30], GP-20, GP-180 [31] and CAPG-GP [26] servers. They tested using SIFT, C-SIFT [32], OpponentSIFT [33], RGB-SIFT, BRISK [34], SURF and found that the model employing SIFT along with VGG-16 showed better balance between precision and recall in detecting the product.

Figure 2.6: Proposed framework of the study

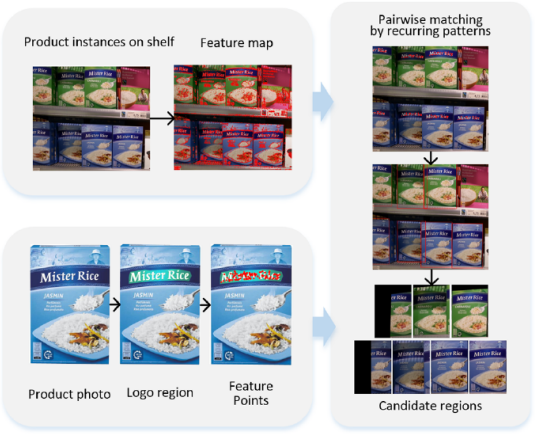


Figure 2.7: Production of candidate region by extracting feature map of the retail items on the shelf and comparing it with the feature points of the training sample

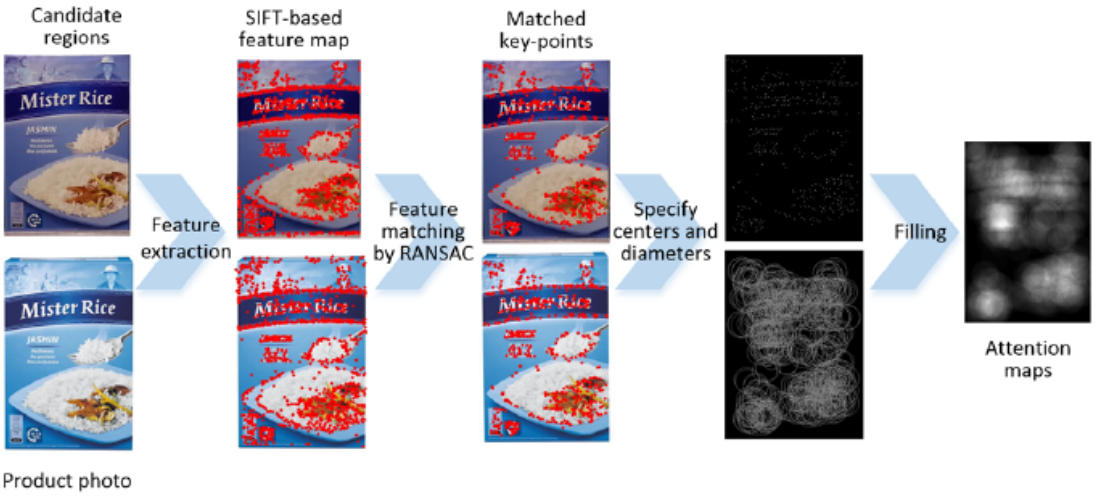


Figure 2.8: Generation of Attention Map Using SIFT

Limitations:

- The system fails to detect retail items when not aligned properly in a front-facing way.
- The system does not perform well on detecting the products, which reflect highlights due to the glossy nature of the packaging.

2.5 Identification of Markers in Challenging Conditions for People with Visual Impairment Using Convolutional Neural Network

This research [35] focuses on employing assistive technology to assist people with visual impairments (PVI) in using markers for indoor navigation. The authors employed CNN to detect markers in this system, which re-defined the identification stage as a classification problem. In the proposed system, they placed printed tags in several areas of interest. The authors were then able to generate a graph with nodes representing the locations of the markers. Their system asks the PVI to

choose a starting point based on the tags in the surrounding area. When the smartphone camera detects a marker, the system uses that marker as a starting point. To begin navigation, the system asks the PVI to use voice instructions to select their destination. Then it looks up the shortest route from this place to the destination using the Dijkstra algorithm.. This path consists of a series of checkpoints that the PVI must travel through in order to reach their destination. When they are navigating, the system provides them with constant guidance as they move from one spot to the next. Tags are utilized to ensure that even if the PVI strays from the intended path, the system can quickly update the current location exactly and continue navigating. According to the authors, the following are the important aspects of this system: 1. How to precisely locate PVI at any time based on the installed tags so that they can continue moving safely. 2. How to use these tags to go from the beginning point to the destination. As a result, the authors split the proposed system into two parts: locating the PVI and guiding them to the destination.

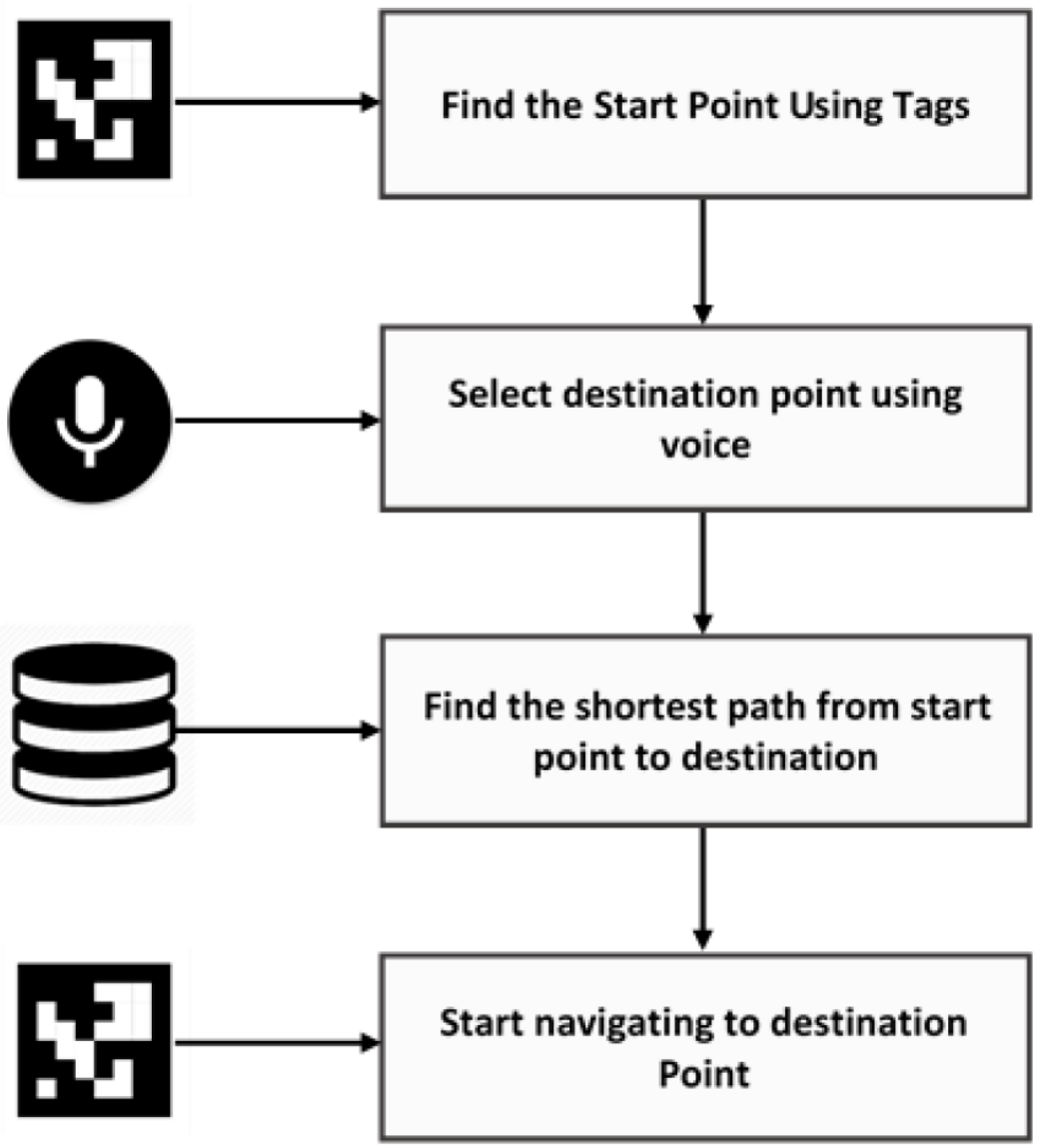
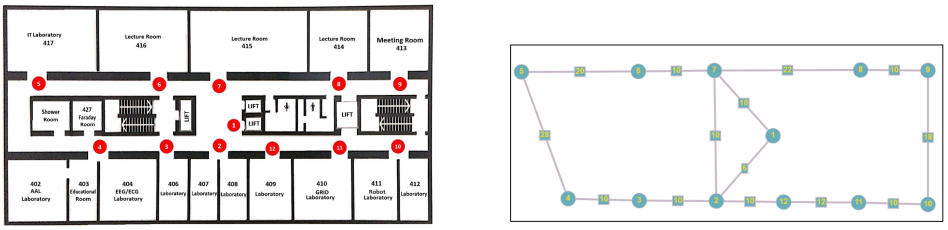


Figure 2.9: Fundamental steps of the proposed system



(a) (b)

Figure 2.10: (a) Floor plan containing markers (in red) at specified location. (b) Graph based on the markers of the same floor plan

The system has two phases: (a) Configuring the premises for visually impaired people by placing markers at the desired locations and creating a map for them (b) Guiding them during navigation using TTS.

The CNN model was trained in batches of 32 and was implemented using the Keras framework. The markers in the synthetic data set were originally 512 × 512 pixels in size, however they were shrunk to 64 × 64 pixels for training the CNN model. The dataset was split into 75-25 for training and validation. The CNN model was trained for 100 epochs in order to obtain all of the experimental results. The authors also used the dataset to train the model in order to determine which marker delivers better outcomes.

Aruco Markers	1 Meter	2 Meters	3 Meters	4 Meters
# of scanned items	Multiple	Multiple	Multiple	Multiple
30 degree	√	√	√	√
60 degree	√	√	√	√
80 degree	√	√	√	√
Moving fast	√	x	x	x
Line of sight	x	x	x	x
Blur	x	x	x	x
Camera out of focus	x	x	x	x
Occlusion	x	x	x	x

QR Codes	1 Meter	2 Meters	3 Meters	4 Meters
# of scanned items	Multiple	Multiple	x	x
30 degree	√	√	x	x
60 degree	√	√	x	x
80 degree	x	x	x	x
Moving fast	x	x	x	x
Line of sight	x	x	x	x
Blur	x	x	x	x
Camera out of focus	x	x	x	x
Occlusion	x	x	x	x

Figure 2.11: Performance evaluation of ArUco Markers and QR codes under different conditions

The authors printed QR codes with a 10 x 10 cm dimension and placed them in the surroundings at regular intervals in the first application. Each QR code contained data about the user's present location. The authors used the OpenCV library for pre-processing. For detecting and identifying QR codes, the authors employed an open-source library called Zxing. When PVI uses this app, the camera is activated and images are captured till a QR code is identified.Instead of QR codes, the authors' second application uses ArUco markers of the same dimension. For detecting and recognizing markers, the authors employed an open-source toolkit called ArUco library. The authors compared the two applications after evaluating them in various situations to determine which one performs better.

After evaluating these two applications under different conditions, the studies found that ArUco markers can be recognized from up to 4 meters away, whereas QR codes are only good for 2 meters. According to the authors, the camera does not need to be in the line of sight of the ArUco markers to detect them from long and short distances. The studies found that QR codes could not be read from more than 2 meters away, regardless of whether the camera was in their line of sight. The authors concluded that Aruco markers are superior to QR codes based on these findings. As a result, the authors decided to use Aruco markers as prototype tags.

CHAPTER 3

THEORETICAL OVERVIEW

In this chapter, we discuss the necessary theoretical overview of the project along with specifications for any hardware tools used.

3.1 Hardware Specification

We use a Logitech 930e webcam to take the live feed of products on the checkout table. Any other standard USB webcam can be used in its place. The specifications of the webcam are listed below [36]:

- Up to 30fps HD 1080p video quality
- 90 Degree Field of view
- ZEISS glass lens
- Autofocus
- 4x Digital Zoom
- Noise cancelling dual mics
- Dimensions Width 3.7 inches (94mm) Height 1.7 inches(43mm) Depth 2.8 inches (71mm)

- Weight 5.7 ounces (162g)
- USB A Cable length - 5ft (1.5m)

3.2 Deep learning and Neural Networks

We consider the biological brain as the most well structured system for processing different information through our sensory organs like our eyes, ears, skin, nose, tongue etc. The brain is made of neurons and the connections between these neurons help process complicated high-level information in the brain. A neuron by itself may be very simple but a complex structure developed from these simple units are responsible for all of the information processing that happens inside a human or any animal.

In the realm of artificial intelligence, the term "artificial intelligence" refers to a computer Neural networks are used to simulate the incredible structure of a neural system that can learn patterns from prior data and gradually improve at the task it is given [37].

Neural networks can vary in complexity from a simple neuron to complex structure of multiple layers of neurons. A neuron is characterized by inputs outputs and weights and activation functions. The inputs are multiplied by the weights, and the sum is sent through a transfer function, resulting in the neuron's output [38]. This simplest form of ANN(Artificial Neural Network) model is called a perceptron [39].

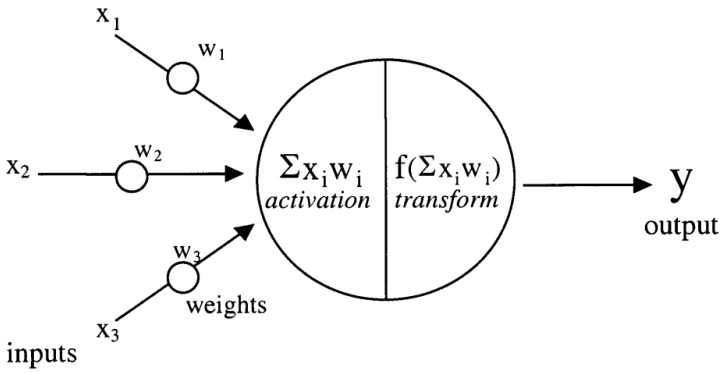


Figure 3.1: Model of Perceptron

Complex structures can be created by connecting neurons in multiple layers. So the input and output layers are separated by multiple hidden layers that help learn complex patterns in the data. The gradient of a feed forward NN is evaluated by backpropagation error and is minimized over time as the network iterates.

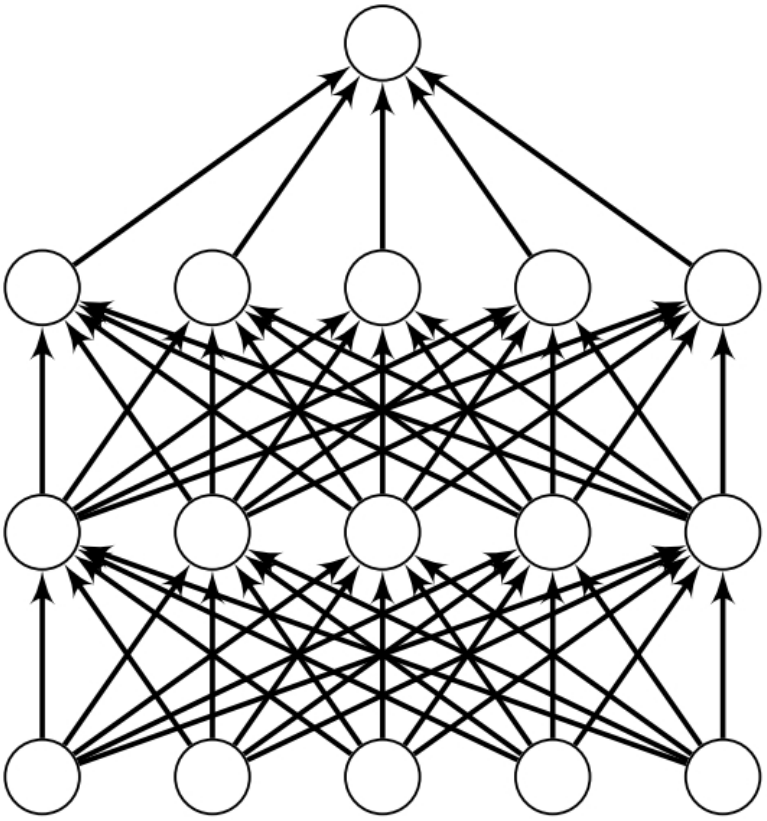


Figure 3.2: Standard Deep Neural Network

The error is slowly minimized through use of optimization algorithms like stochastic gradient descent. This leads the model to become more accurate in predicting or classifying the problem set it is designed for [40].

3.3 Transfer Learning

With an abundance of machine learning algorithms focused on training a specific work, it is more efficient to take the knowledge of processes for completing a single task, and use it to learn similar work. This process of using the knowledge of completing a task to help learn to complete a different task is Transfer Learning. Most of the deep learning today is used for completing an isolated work. However, building a new algorithm for a new task is not only time consuming but also inefficient since this process needs to run through a lot of trial and error. By using transfer learning we can accelerate the speed of learning at an efficient rate.

Some of the works for transfer learning are included in neural networks, Bayesian networks, Markov Logic Networks, Q-learning and policy search [41]. While it seems like using training data and future data in the same field of work might be more relative, using transfer learning to train a dataset on a completely different classification task's knowledge can be impressively improving in the context of performances. In real-world problems, algorithms for each different task might not work as well as it would in the case of implementing a transfer learning on that task[1].

Transfer learning can improve the learning process significantly from three contexts:

- The initial learning rate can greatly improve by using transfer learning. While a blind algorithm with no knowledge of completing a task starts off by randomly mathematical approaches, a transfer learning method can help accelerate the slope of initial learning that has already worked on a different task before.

- The saturation point of a learning algorithm reaches much earlier for transfer learning, with a higher performance too. While the amount of time to learn from scratch is significantly longer for learning it from scratch.
- Multi-Task learning is a process where different tasks share knowledge with each other. In this category, information can flow freely between tasks making it easier and faster for them to learn.

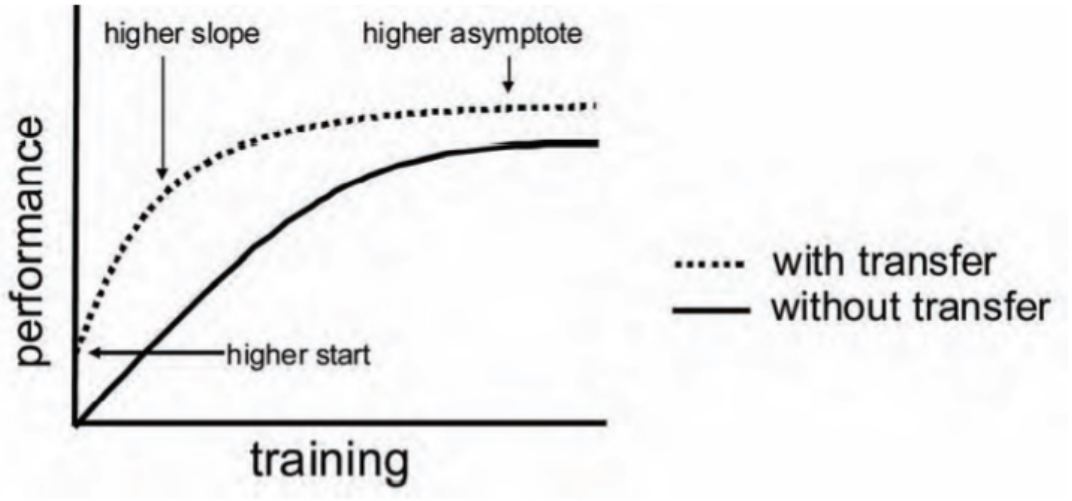


Figure 3.3: Improvements due to transfer learning [1]

3.4 Defining Object Detection

In the field of computer vision (CV) the term "Object Recognition" is used to generalize an array of tasks that are related to identifying objects in an image. There are tasks such as Image classification, Object localization and segmentation that fall in this category of tasks.

Object detection has been in the research field for a very long time. The first era of object detection leaned on cold weapon detection, where most of the features were handcrafted. Around 2012, first CNN based object detection was used [42][43] and since then have constantly evolved. Today we use advanced DNN for object detection.

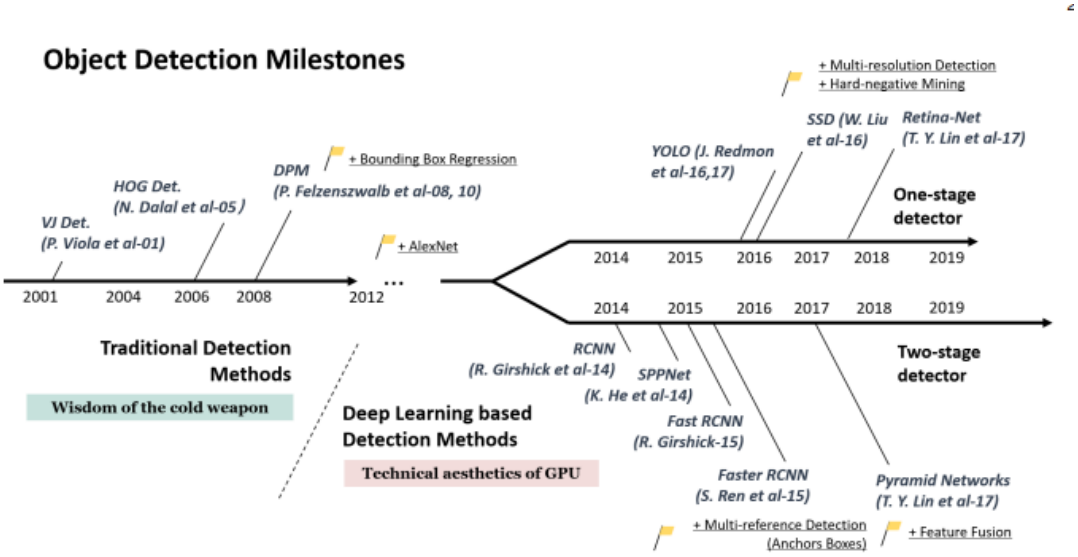


Figure 3.4: Roadmap of Object Detection [2]

Using DNN in object detection has shown incredible performance in object detection. For detecting an object we take an image, run it through specified layers of DNN, which then produces masks for the object along with the portions of the object. This process localizes the objects location precisely [44].

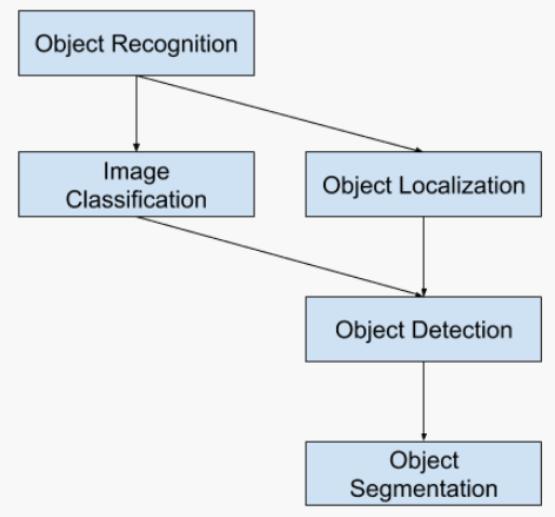


Figure 3.5: Core Tasks of object detection

The core tasks of object detection can be portrayed below tasks:

- Image classification refers to the task of identifying what the object in an image is or in other words which predefined class does this object belong to.
- Object Localization is the task of locating where the objects in an image are. This is most commonly done by drawing bounding boxes around them.
- Object Detection is a combination of both these tasks, it first localizes the objects in an image and then identifies which class each object belongs to.
- Object Detection is a combination of both these tasks, it first localizes the objects in an image and then identifies which class each object belongs to.
- Object Segmentation / Semantic Segmentation is an extension of object detection. Instead of drawing bounding boxes around an object, it colours the pixels encompassing each object in a unique colour, visually separating them from each other [45].

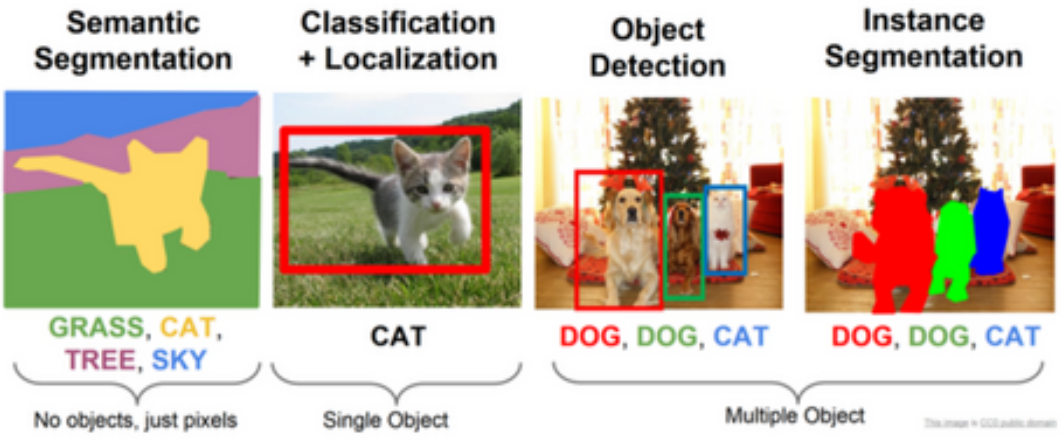


Figure 3.6: Comparison of semantic segmentation, classification and localization, object detection and instance segmentation. [3]

3.5 Object Detection Algorithms

3.5.1 R-CNN

R-CNN stands for region-based convolutional neural network which is a neural network specifically used in machine learning for the purpose of object tracking. Computer vision has seen rapid growth in technological aspects largely due to baseline systems such as fast/faster R-CNN [46]. It is known to provide state-of-the-art results in the field of object tracking. Mask R-CNN architecture also exists as an intuitive extension of fast R-CNN which adds a branch for each Region of Interest, forecasting segmentation masks [47].

The primary purpose of R-CNN is to identify the primary objects of any given picture. The detection process is made up of three different modules. The first creates proposals based on independent regions. The second module is a feature extractor implemented through deep CNN. The third module is for classification, and it may follow any classification algorithm such as kNN, SVM etc. [48]. The R-CNN pipeline proposes a set number of boxes per picture that are most likely to contain the target items. All proposal boxes (even small ones) are enlarged to a canonical size at the last pooling layer, which means that a complete feature map is created for each proposal box [49].

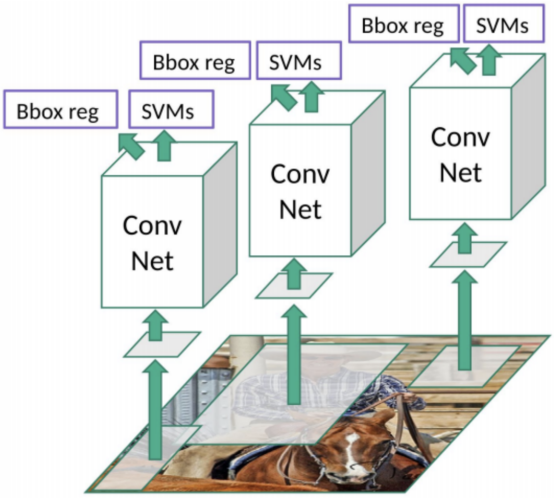


Figure 3.7: R-CNN Structure RCNN

The advantage of R-CNN is that it is highly accurate, but it is also prolonged and unsuitable for real-time applications. The difference in test times is shown in the following table. Fast and Faster R-CNN works much better for applications where real-time input is involved.

Table 3.1: Comparison of speed between R-CNN, Fast R-CNN and Faster R-CNN

	R-CNN	FastR-CNN	FasterR-CNN
Test time perimage(with Proposal)	50 sec	2 sec	0.2 sec
(Speedup)	1x	25x	250x
mAP(VOC 2007)	66.0	66.9	66.9

3.5.2 SPP Net

The SPPnet technique creates a convolutional feature map for the whole input picture and then uses a feature vector taken from the shared feature map to classify each object proposition [50]. At test time, SPPnet speeds R-CNN by 10 to 100 times. Due to quicker proposal feature extraction, training time is also decreased by three times. However, SPPNEt exhibits similar problems as R-CNN. Extracting features, fine tuning a network using log loss, training SVMs, and ultimately fitting bounding-box regressors are all part of the training workflow. However, it is not similar in that the convolutional layers that precede the spatial pyramid pooling cannot be updated using the finetuning technique. This limits the precision of very deep neural networks [51].

3.5.3 Fast RCNN

Although RCNN is highly popular for object tracking, it is still far too slow for real-time applications. Fast RCNN is known to have much better performance with minimal delay. The primary issues of using RCNN are:

1. R-CNN uses log loss to finetune a ConvNet on object suggestions. The SVMs are then fitted to ConvNet features. These act as object detectors,

and they replace the classifier learnt by fine tuning. Finally, in the last stage, regression of the bounding boxes is discovered. This means there are multiple stage pipelines to go through before the data can be processed.

2. There are multiple object proposals based on a region, and each has to be extracted in order for R-CNN to work. Considering the high dimension of modern data it would take a very long time and a lot of GPU memory to allow the CNN to work properly.

3. Detection of the average of a VGG16 image takes 47 seconds. This means the detection process is very slow.

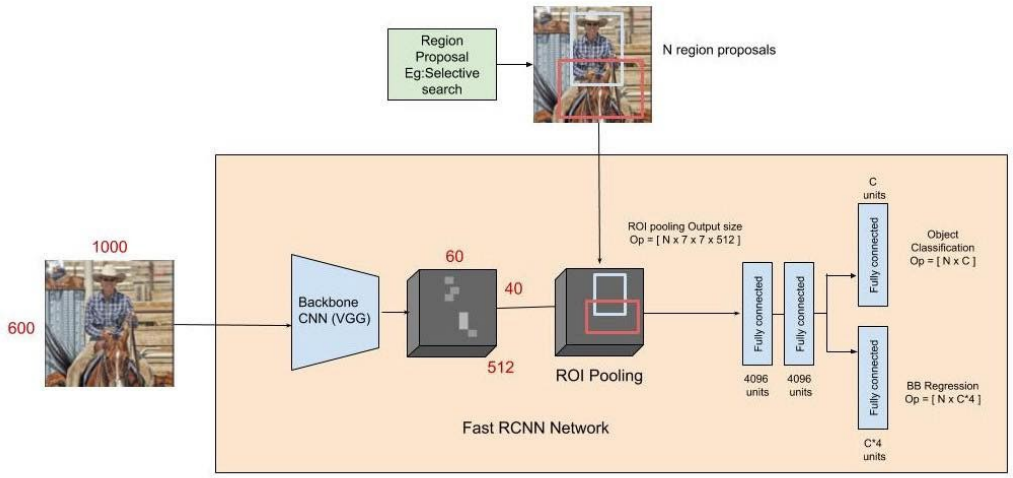


Figure 3.8: Faster RCNN architecture

A full picture and a collection of item suggestions are fed into a Fast R-CNN network. To create a conv feature map, the network first analyzes the entire image with multiple convolutional (conv) and max-pooling layers. After that, a region of interest (RoI) pooling layer generates a fixed-length feature vector from the feature map for each object proposition.

3.6 YOLO Overview

3.6.1 History

The YOLO (You Only Look Once) algorithm was introduced in 2015 by researcher Joseph Redmon and others using a novel approach, treating object detection as a regression problem using only a single neural network. It brought remarkable achievements in the field of object detection than other models could. Five different versions of YOLO have been developed hitherto, of which the first three were developed by the original authors, Joseph Redmond and others. However, he discontinued his research in the computer vision field after the release of version three for ethical reasons. In 2020, Alexey Bochkovskiy, who developed the previous three versions of YOLO with Redmond, released version 4, YOLOv4 [52], on the official YOLO Github account. Glenn Jocher and his Ultralytics LLC research team developed the YOLO algorithm using the Pytorch framework and released the YOLOv5 only after one month of the release of YOLOv4 with a few changes and improvements. Although none of the original authors was involved in developing this version, the YOLOv5 outperforms all the previous versions in terms of both accuracy and speed. Since no paper exists yet of YOLOv5, there is controversy in the research community if this model justifies the original branding of YOLO [53].

Before YOLO, the other CNN models like R-CNN employed RPPNS(Regional Proposals Networks), which was an iterative method for classifying different regions of the input image. For each region, the model produced a bounding box, ran a classifier on the box, applied post-processing and refined the bounding box. Optimizing detection performance on individual stages of the networks separately made it very difficult [54].

YOLO Redmon_2016_CVPR [4] came up with an approach to unify all stages on a single neural network. The entire detection pipeline of YOLO consists of a single network facilitating the network to be optimized for detection performance easily from end to end. By anticipating the bounding boxes and class probabilities, the network directly analyzes the image and recognizes not only what items are present but also where they are located inside the image. Rather than using an iterative

method, the system can make predictions for all of the objects in the image at once, hence the name YOLO (You Only Look Once). Even though it sometimes struggles with smaller objects, YOLO being orders of magnitude faster than other object detection algorithms mentioned above is our reason for choosing it for the project. [55]

3.6.2 How it works

YOLOv1 applies grid cells of size S x S (7×7 by default) on an image. The grid cell in which the object's center is located is responsible for detecting that object. Despite the fact that the object appears on other grid cells, those cells are discarded as the center isn't present inside them.

Each grid cell predicts B bounding boxes, each of which consists of some parameters (center, width, height) and confidence score. The confidence score is used to determine if an object is present or absent in that specific bounding box. The confidence score can be calculated as:

$$\text{confidence score } = p(\text{ Object }) * IOU_{\text{truth}}^{\text{pred}} \tag{3.1}$$

Where p(Object) is the probability of the presence of an object inside the cell and $\text{IOU}_{pred}^{truth}$ is the intersection over union of the predicted bounding box and the ground truth bounding box. If there is no object in a cell, then p(Object) $\rightarrow$0 hence the confidence score is also close to 0. Otherwise, the score is close to $\text{IOU}_{pred}^{truth}$.

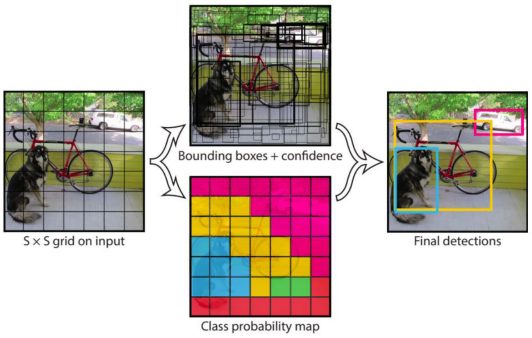


Figure 3.9: YOLO model with 7*7 grid was applied to input Image(Redmon, et al. 2016) [4]

Along with the confidence score, each bounding box employs 4 other parameters, which are x, y, w and h. The center coordinate is represented by the x,y parameters, width by the w parameter and height by the h parameter. Therefore, each bounding box uses a total of 5 parameters: x, y, w, h and the confidence score.

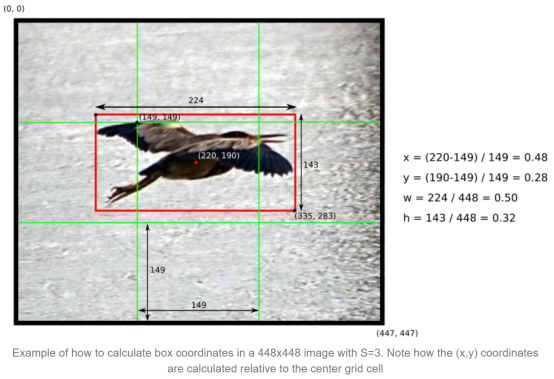


Figure 3.10: Example of how to calculate box coordinates in a 448*448 image with S=3. [5]

To predict in which class the object belongs to, conditional probability is applied on the grid cell given that the cell contains one object, Pr(Class(i) — Object). Therefore, if the grid cell contains no object, the loss function will not make an incorrect class prediction. Regardless of the number of bounding boxes B, the network only predicts one set of class probabilities per cell. In total, there are S $\times$ S $\times$ C class probabilities and an S $\times$ S $\times$ (B * 5 + C) tensor as output which is obtained by adding the class predictions to the output vector.

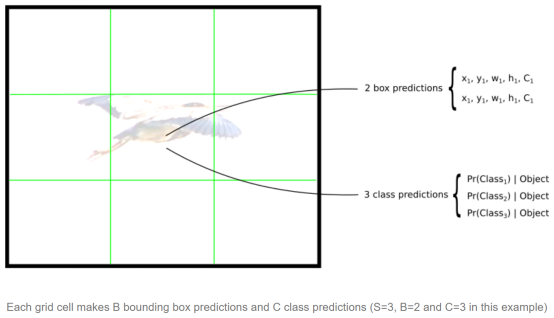


Figure 3.11: Each grid cell makes B bounding box predictions and C class predictions. [5]

The predicted bounding box vectors are denoted by output vector $\hat{y}$ and ground truth bounding box vectors are denoted by vector label y. Vector label y and

predicted vector $\hat{y}$ could be indicated in the figure as shown below. There is no object present in the purple cell, hence the confidence score of bounding boxes in this cell is equal to 0, which leads to discarding all the remaining parameters.

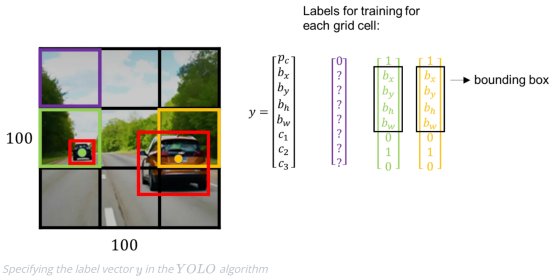


Figure 3.12: Matrix calculation for YOLO [6]

Lastly, YOLO removes all the bounding boxes which do not contain any object like the purple cell in the above figure, or contain the same object as other bounding boxes by using Non-Maximum Suppression (NMS). By setting a threshold value, NMS removes all the overlapping bounding boxes which have intersection over union (IOU) value higher than the threshold value [56].

3.6.3 YOLO Overview

YOLOv1 Architecture

The YOLO model was created by the original authors to include Darknet architecture, which analyses all image features, followed by two fully connected layers that perform bounding box prediction for objects. The authors used S=7, B=2, and C=20 in the Pascal VOC dataset to test this model for which the final feature maps are 7×7, and the output size was $(7 \times 7 \times (2 \ \ 5 + 20))$ [57].

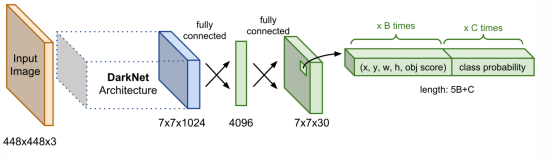


Figure 3.13: Network architecture of YOLO [7]

For uncomplicated datasets, the authors introduced the Fast-YOLO model with 9 CNN layers in Darknet architecture, while the normal-YOLO model with 24 CNN layers in Darknet design can handle more complex datasets and produce higher accuracy. Instead of Leaky Rectified Linear Unit (leaky ReLU) activation, the final layer utilizes a Linear activation function [58].

$$\phi(x) = \begin{cases} x, \text{ if } x > 0 \\ 0.1x, \text{ otherwise} \end{cases} \tag{3.2}$$

Name	Filters	Output Dimension
Conv 1	7 x 7 x 64, stride=2	224 x 224 x 64
Max Pool 1	2 x 2, stride=2	112 x 112 x 64
Conv 2	3 x 3 x 192	112 x 112 x 192
Max Pool 2	2 x 2, stride=2	56 x 56 x 192
Conv 3	1 x 1 x 128	56 x 56 x 128
Conv 4	3 x 3 x 256	56 x 56 x 256
Conv 5	1 x 1 x 256	56 x 56 x 256
Conv 6	1 x 1 x 512	56 x 56 x 512
Max Pool 3	2 x 2, stride=2	28 x 28 x 512
Conv 7	1 x 1 x 256	28 x 28 x 256
Conv 8	3 x 3 x 512	28 x 28 x 512
Conv 9	1 x 1 x 256	28 x 28 x 256
Conv 10	3 x 3 x 512	28 x 28 x 512
Conv 11	1 x 1 x 256	28 x 28 x 256
Conv 12	3 x 3 x 512	28 x 28 x 512
Conv 13	1 x 1 x 256	28 x 28 x 256
Conv 14	3 x 3 x 512	28 x 28 x 512
Conv 15	1 x 1 x 512	28 x 28 x 512
Conv 16	3 x 3 x 1024	28 x 28 x 1024
Max Pool 4	2 x 2, stride=2	14 x 14 x 1024
Conv 17	1 x 1 x 512	14 x 14 x 512
Conv 18	3 x 3 x 1024	14 x 14 x 1024
Conv 19	1 x 1 x 512	14 x 14 x 512
Conv 20	3 x 3 x 1024	14 x 14 x 1024
Conv 21	3 x 3 x 1024	14 x 14 x 1024
Conv 22	3 x 3 x 1024, stride=2	7 x 7 x 1024
Conv 23	3 x 3 x 1024	7 x 7 x 1024
Conv 24	3 x 3 x 1024	7 x 7 x 1024
FC 1	-	4096
FC 2	-	7 x 7 x 30 (1470)

Figure 3.14: YOLOv1 model architecture. [5]

Loss function

The backbone of YOLO's loss function is the sum-squared error. Multiple grid cells containing no objects have a confidence score of zero, for which the gradients of cells that contain the objects gets overwhelmed. To reduce training divergence and model instability, YOLO applies the largest penalty for predictions from bounding boxes containing objects (λ_{coord}=5) and the lowest penalty for predictions from bounding boxes with no object (λ_{noobj}=0.5) [59]. The loss function of YOLO is calculated by adding the loss functions of all bounding box parameters:

$$
\begin{aligned}
\mathcal{L} = & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
& + \lambda_{\text{coord}} \sum_{i=0}^{s^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left(c_i - \hat{C}_i \right)^2 + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} \left(C_i - \hat{C}_i \right)^2 \\
& + \sum_{i=0}^{s^2} \mathbb{1}_{i}^{obj} \sum_{c \in \text{ classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2
\end{aligned}
\tag{3.3}
$$

Batch Normalization

YOLOv2 implemented batch normalization as a normalization method that facilitates faster and more stable deep neural network training by stabilizing the input layer distribution [60]. This method normalizes the output of each layer after activation to a zero-mean state with a standard deviation of 1.

Mini $-$ batch mean: $\mu = \frac{1}{m} \sum_{i=1}^{m} z^{(i)}$

Mini $-$ batch variance: $\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} \left(z^{(i)} - \mu \right)^2$

Normalize: $z_{\text{norm}}^{(i)} = \frac{Z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$

Scale and shift: $\tilde{z}^{(i)} = \gamma z_{\text{lorm}}^{(i)} + \beta$

This technique decreases training time while simultaneously improving to regularize the network. The network also does not need to use dropout anymore to avoid

overfitting. Batch normalization improved mAP(mean average precision) by more than 2% in YOLOv2 [61].

High Resolution Classifier

In YOLOv1, the model trains the feature extractor(classifier network) at 224 × 224 and enhances the detection resolution to 448 × 448 causing the network to simultaneously switch between learning object detection and adapt the new input resolution. Whereas in YOLOv2, the feature extractor is first processed on ImageNet [62] at the full 448 × 448 resolution for 10 epochs, which allows the model extra time to modify its filters to perform better on higher resolution input than YOLOv1. Then the resulting network is trained on detection which increases mAP (Mean Average Precision) by almost 4% .

Bigger network with ResNet

YOLOv2 failed to detect small objects because the input image lost its fine-grained details due to the downsampling done before entering the deeper layers. ResNet (Residual networks) allowed the activations to enter deeper layers without losing details of input by implementing the concept of skip connections [63].

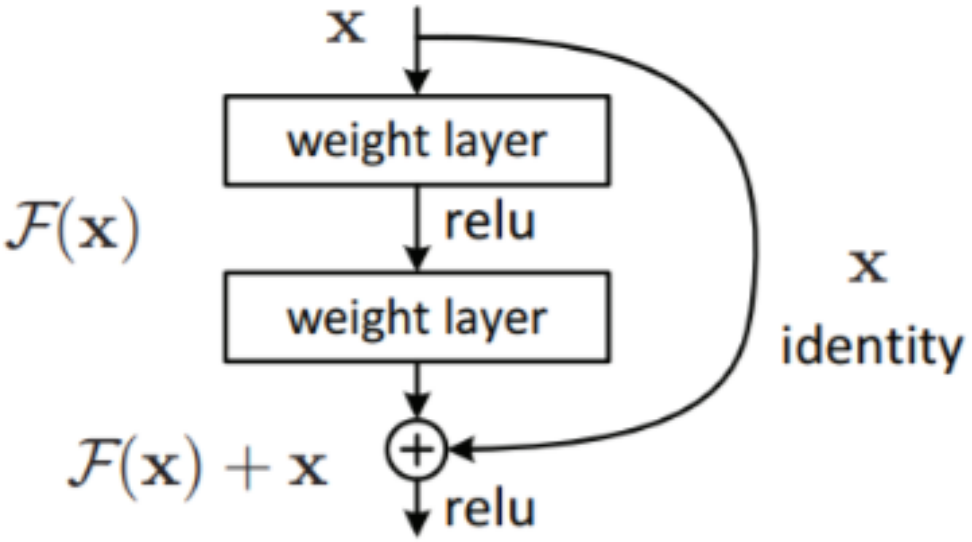


Figure 3.15: Building block of residual learning.

Feature Extractor The architecture of YOLOv3 proposes a hybrid feature extractor combining YOLOv2, Darknet-53 and ResNet [64]

	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 3.16: Darknet-53.

3.6.4 YOLOv5

The previous versions of YOLO were developed on the Darknet framework, which is written in the C language. However, YOLOv5 is developed in PyTorch, which is written in the Python language. As YOLOv4 and YOLOv5 are developed using

two different languages on two different frameworks, their performance cannot be compared with accuracy. In certain aspects, YOLOv5 has shown better performance than YOLOv4 and has partly gained recognition in the research community besides the other versions of YOLO. Since YOLOv5 is written in the Python language, many applications will be easier to implement.

The YOLOv5 model can be summarized as follows:

Backbone: Focus structure, CSP network

Neck: SPP block, PANet

Head: YOLOv3 head implementing GIoU-loss

CSP Backbone

Both YOLOv4 and YOLOv5 have employed the CSP Bottleneck [65] in their architecture. The CSP addresses the problem of the duplicate gradient of other convolutional networks' backbones, which is crucial for the YOLO architecture as inference speed is extremely important for the YOLO model.

The CSP networks (Cross Stage Partial Network) are based on DenseNet [66] DenseNet facilitates solving the vanishing gradient problem and reducing the network parameters enabling the YOLOv5 architecture to have a small model size.

Adaptive anchor boxes

The authors of YOLOv2 proposed the concept of anchor box as well as a method for selecting anchor boxes that are similar in size and shape to the ground truth bounding boxes in the training samples. They chose the 5 best-fit anchor boxes for the COCO dataset of 80 classes using the k-mean clustering algorithm and set them as default. However, when a new dataset emerges with a class that is not one of the 80 classes in the COCO dataset, the default anchor boxes are unable to immediately adjust to the ground truth bounding boxes. A giraffe, for instance, requires a thinner and higher anchor box than a square box. Therefore, the k-mean clustering algorithm is used first to get the best-fit anchor box for the giraffe dataset.

In YOLOv5, the anchor box selection process has been integrated. Therefore, this network does not need to consider any of the datasets to be used as input.It automatically learns the best-fit anchor boxes for any type of dataset and uses them during training.[67]

Easy to Use

YOLOv5 is easy to use [68] for computer vision-related applications compared to other object detection architectures since it is written in Python language. Only Pytorch and some lightweight python libraries need to be installed to implement this model. This model can be trained fast, which reduces experimentation costs for the developers.

3.7 Data Augmentation

Deep learning models need a lot of data to learn properly and the acquisition of such large volumes of data are not always possible. With smaller datasets probability of overfitting increases and if we look at the most effective models, we can see that they are clearly driven by the largest datasets. The lack of good quality data can be circumvented by using data augmentation in order to create new data [69].There are several ways for image based data augmentations,

3.7.1 Basic Augmentation Techniques

Geometric Augmentations:

This includes easy to perform actions like rotation, flipping, translation, random cropping. These methods might be easy to perform, but they sometimes are not label-preserving in nature. So the images have to be relabeled post augmentation.

Colour based Transformation:

Changing the colour spaces for images by manipulating its colour channels in an augmentation technique employed to tackle the lighting based biases in most image recognition problems [70].

3.7.2 Synthetic Data

One of the most important issues in computer vision problems is generating a labelled dataset. Most of the time esp. in case of segmentation tasks the data has to be manually labeled in order to produce good quality results. It is a very tedious process to acquire and label a huge amount of image data manually, so an alternative solution is generating synthetic data [9].There are many methods for synthetic generation:

Mixing Images:

Randomly mixing different parts of images or changing background based on a set of objects and background images are very simple ways of synthetic data. Images can be randomly cropped and added to other images.

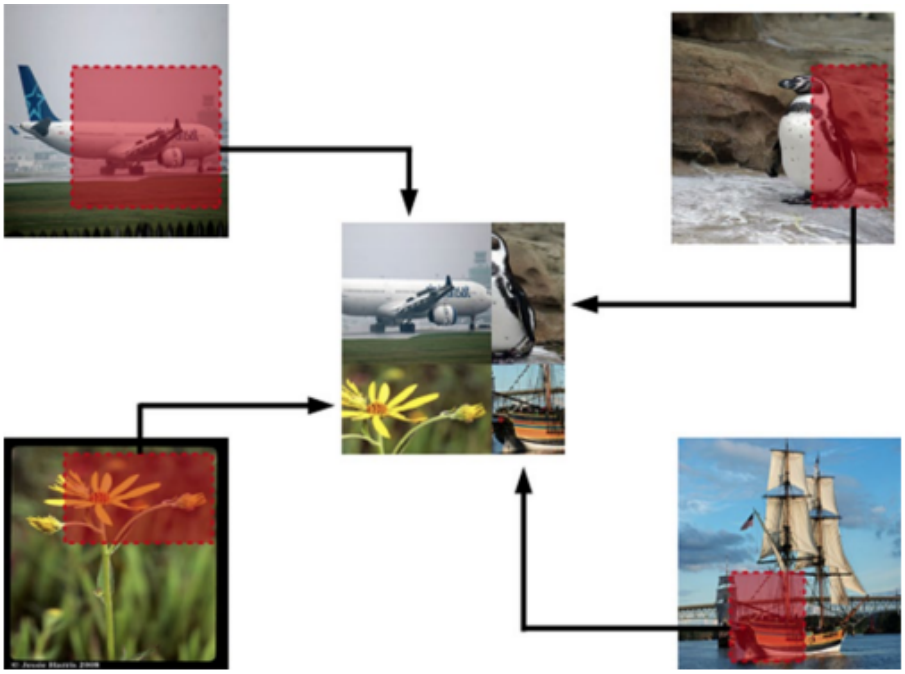


Figure 3.17: Image mixing via random cropping.[8]

Another technique for synthetic data generation is composing images by masking objects on different backgrounds. Geometric augmentation techniques can also be applied alongside this method to generate more effective synthetic data [71].

Figure 3.18: Image composition via masking

Generative Adversarial Networks(GAN):

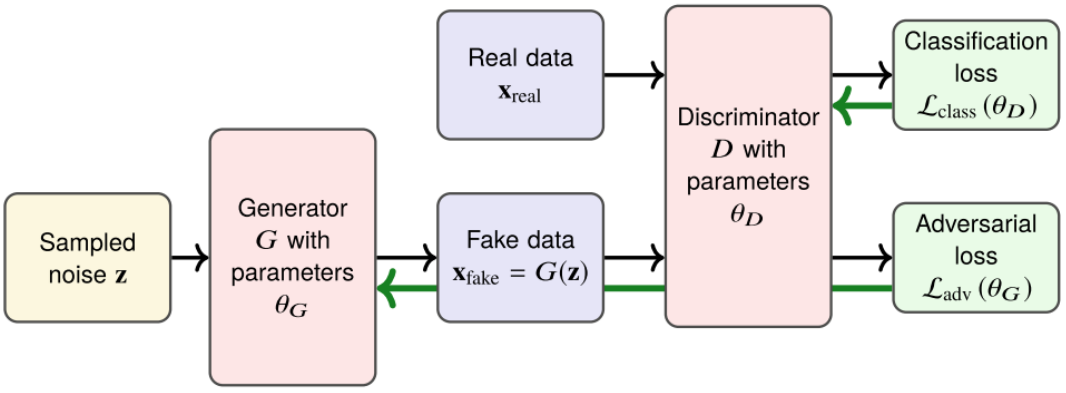


Figure 3.19: Basic GAN Architecture [9]

GAN is a framework of two models in an adversarial setting where one model tries to create an image of a sample similar to something from the training set but another discriminative model tries to predict if the generated sample came from the training set or not. Both of the models are trained together with the purpose of making the generative network create entirely new samples of an object from the training data that are significantly different.

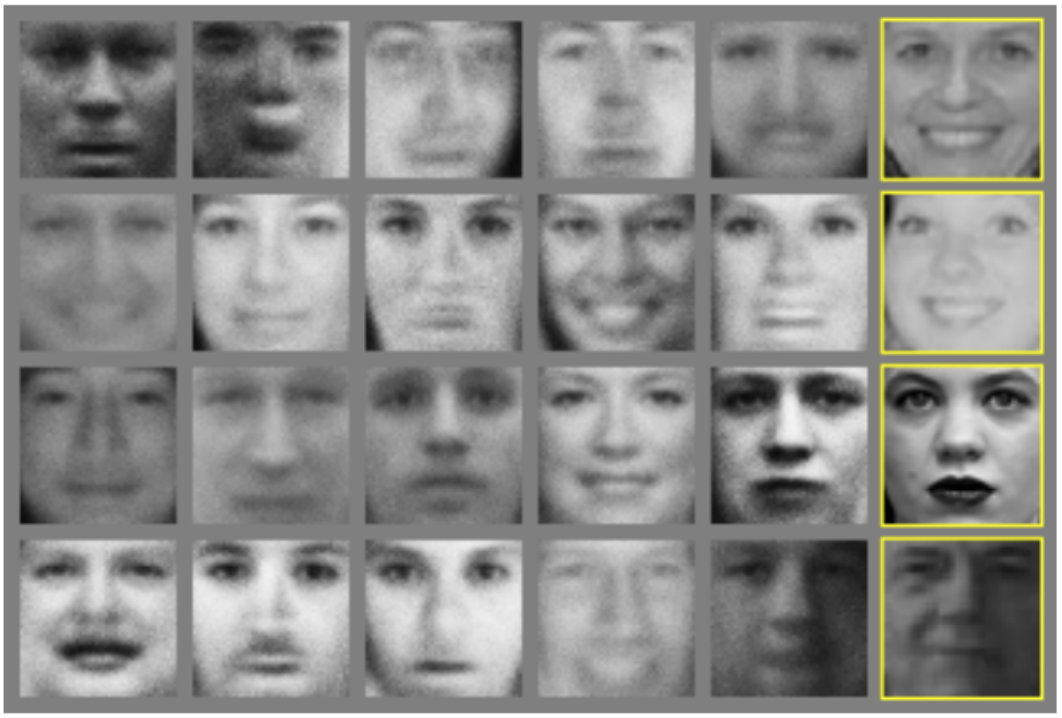


Figure 3.20: Synthetic image generation with GAN (Yellow marked column is the closest training sample to the neighbouring output.)

3.8 YOLO Annotation

For Object detection algorithms, annotation means identifying the location and class of objects present in an image. This can be done in several different ways, such as bounding boxes, polygonal segmentation, semantic segmentation, 3d cuboids, etc. Bounding boxes are the most common type of annotation used in Computer Vision. An object in an image is identified by the coordinates of a box encompassing its visible area [72].

YOLO has a specific format for annotations. . YOLO uses the following way to annotate objects in an image. Each object's bounding box must be represented with its class id, center coordinates, height, and width. The coordinates and dimensions must be normalized by the dimensions of the image. We create a .txt

file with the same name as the image file for each image with the contents like this:

cls_id norm_x_center norm_y_center norm_width norm_height

$$norm - x - center = \frac{x - center}{img - width}$$

$$norm - y - center = \frac{y - center}{img - height}$$

$$norm - width = \frac{width}{img - width}$$

$$norm - height = \frac{width}{img - height}$$

Figure 3.21: Bounding Box

In the Figure we can see the center of coordinates of the bounding box (198.98, 204.01) and the height and width of the bounding box are 120.64px and 333.84px. The image is 416x416. In YOLO's annotation format we need to normalize this values with respect to the dimensions of the image.

$$norm - x - center = \frac{198.98}{416} = 0.4639$$

$$norm - y - center = \frac{204.01}{416} = 0.4904$$

$$norm - width = \frac{120.64}{416} = 0.29$$

$$norm - height = \frac{333.84}{416} = 0.8025$$

Let's assume that the class for this product is 1, So the text file of this image will contain: 1 0.4639 0.4904 0.29 0.8025

3.9 Optimizers

Optimizers are algorithms that work on hyperparameters, the values of which determine the behaviour of the optimizer. In a network, the optimizer determines its learning slope for achieving the intended value of the loss function.

For a loss function produced over a neural network l, its first derivative can be taken as a gradient (theta) can be represented as the hyperparameters. For the above scenario, an optimizer is an algorithm that iterates to reach a point for which l will be minimized. For different optimizer algorithms, the loss function and hyperparameters can differ. [73]

3.9.1 SGD (Stochastic gradient descent)

SGD is a fairly simple nut efficient optimizer. The mathematical idea behind this optimizer is to follow the steepest slope to reach the localized minima.

For a neural network that needs to produce a value for the hyperparameter which reduces the loss function to a minimum, SGD approaches the algorithm with below relation:

$$w_k = w_{k-1} - \alpha_{k-1}\hat{\nabla} f\left(w_{k-1}\right) \tag{3.4}$$

Where, w_k denotes the k^{th} iterate, α_k is a(tuned) step size sequence, also called as the learning rate.

With f being the loss function. SGDM, a variant of SGD, uses inertia to calculate the learning rate of the optimizer. This can greatly affect the performance of the optimizer. SVGM uses the following iteration to reach a minimum loss point:

$$\begin{aligned} v_k &= \beta v_{k-1} + \hat{\nabla} f\left(w_{k-1}\right) \\ w_k &= w_{k-1} - \alpha_{k-1} v_k \end{aligned} \tag{3.5}$$

where

$$\beta \in [0, 1) \tag{3.6}$$

is a momentum parameter ans v_0 is initialized to 0. [74]

However SGD may still reach a local minima or a saddle point and its learning rate must be set in between a range to get the minima, or divergence might occur. [75]

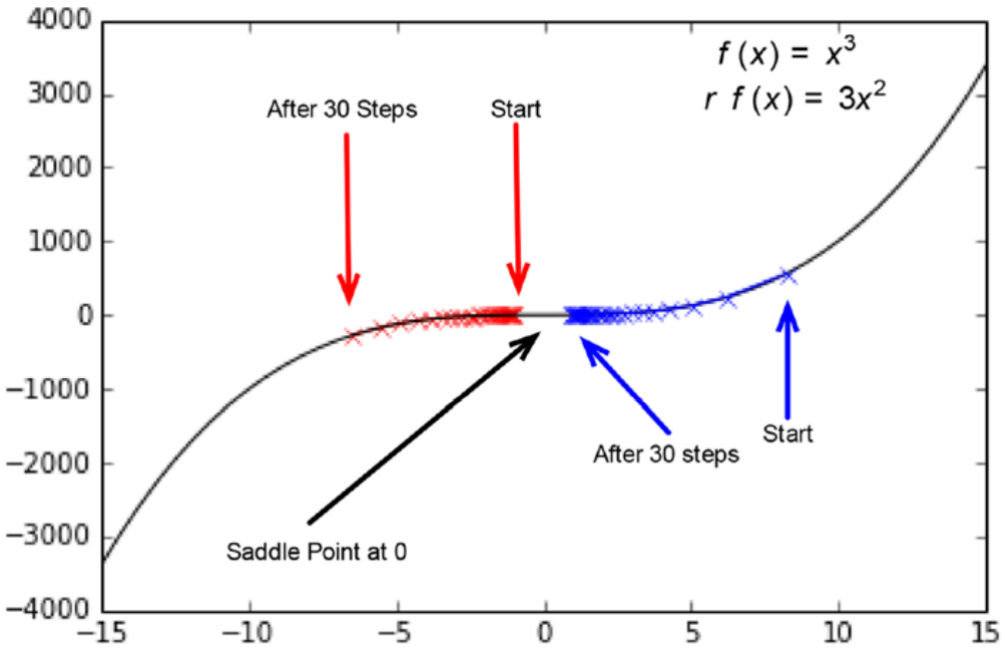


Figure 3.22: Local Minima for SVG algorithm

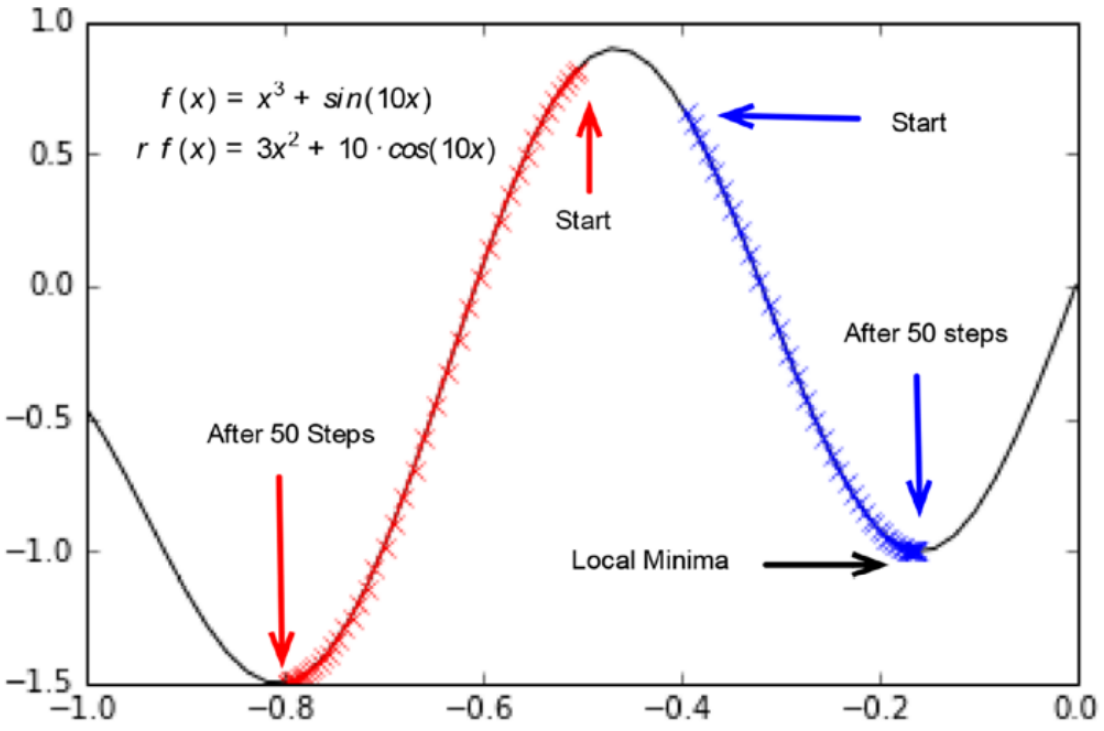


Figure 3.23: Saddle Point in minimizing the loss

3.9.2 ADAM

With an attempt to correct these shortcomings, several adaptive optimizers were developed, namely ADAM, AdaGrad, RMSProp. With AdaGrad and RMSProp, the initialization makes the earlier updates messy and can tilt the scaling parameters. ADAM tired to solve this problem using below iteration: $w_k = w_{k-1} - \alpha_{k-1} \cdot \frac{\sqrt{1-\beta_2^k}}{1-\beta_1^k} \cdot \frac{m_{k-1}}{\sqrt{v_{k-1}}+\epsilon}$, where

$$m_{k-1} = \beta_1 m_{k-2} + (1-\beta_1)\,\hat{\nabla} f\,(w_{k-1})$$
$$v_{k-1} = \beta_2 v_{k-2} + (1-\beta_2)\,\hat{\nabla} f\,(w_{k-1})^2$$

[74]

In recent studies, ADAM has shown exceptional performance with its adaptive ability. With the generalization and improved training performances, ADAM has solved the problem of inability of SGD to adapt to different scales. [72]

3.10 YOLO Hyperparameters

Hyperparameters are parameters of a machine learning model that are external and cannot be calculated from the data it is operating on. A model tries to minimize some sort of loss function. This process involves some parameters to be kept fixed during the process. Changing these parameters will lead to changes in the behaviour of a model. These parameters, which have to be manually adjusted by humans, are called hyperparameters [76] [77].

YOLOv5 has several hyperparameters; they are as follows [78]:

- Initial learning Rate (lr0)
- Final onecycle LR learning rate (lrf)
- SGD momentum (momentum)
- Weight decay (weight_decay)
- warmup -epochs
- Warmup momentum
- Warmup initial bias
- Box loss gain (box)
- Class loss gain (cls)
- Class BCEloss positive weight (obj_pw)
- Object loss gain (obj)
- Object BCEloss positive weight (cls_pw)
- IoU training theshhold (IoU_t)
- Anchor multiple threshold (anchor_t)
- Focal loss gamma (f1-gamma)
- Images HSV-Hue , Saturation and Value augmentations (hsv_h, hsv_s, hsv_v)

- Image translation (translate)
- Image scale (scale)
- Image shear(shear)
- Image perspective
- Image flip up-down (flipud)
- Image mosaic(mosiac)
- Image mixup(mixup)
- Segment copy-paste (copy_paste)

There are many methods of setting these hyperparameters like grid search and random guessing. Here we will talk about a different approach to hyperparameter tuning, which is hyperparameter evolution.

3.11 Hyperparameter Evolution

Hyperparameter evolution is a Genetic algorithm-based approach to hyperparameter tuning. Genetic algorithm is a machine learning algorithm based on the concepts of natural selection. It is basically a probability-based search function that searches the optimal solution by mixing and matching parameters through multiple generations. The search begins by generating an initial population based on some given parameter. The chromosome or gene pool of the algorithm contain numbers that are a possible solution, new generations have created either crossover through mating or changed through mutation in order to go from one generation to another. Fitness is a function that helps define the optimal solution the algorithm is searching for. Fitness is evaluated on all members of each generation and only the ones with higher fitness are taken as the basis for creating the next generation via crossover or random mutation.[79]

In case of Hyperparameter tuning, the chromosomes are formed with hyperparameters of a model as genes.

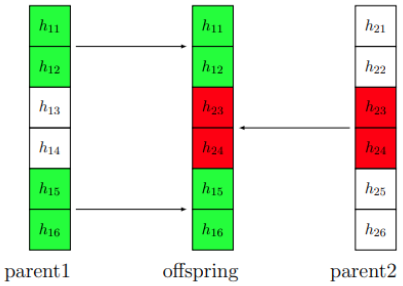


Figure 3.24: Hyperparameter evolution with GA

Some fitness function is determined based on the metrics of the task the model is meant to perform. Fitness is tested and new generations are created based on the GA. New generations are created using crossover of the highest-scoring chromosomes of the last generation. The mutation rate of the GA shouldn't be high as it may cause it to behave like an unintelligent random search function [80]. Based on the value of fitness scores the algorithm ends up with optimized values for all hyperparameters of the model.

3.12 QR Code: How It Works

In general, a QR code functions similarly to a barcode at grocery shops. It's a machine-readable image that can be read quickly using a camera. A QR code is made up of a set of black squares and dots that represent certain types of data. When the camera scans this code, it gets translated into human understandable information. The different parts of a QR code which represents different kinds of data are discussed below. [81]

3.12.1 Position Detection Markers

Figure 3.25: Position Detection Markers

These are located in three corners of each code and helps a scanner to precisely identify and read the Code quickly while also displaying the orientation in which the Code is printed. They help to determine the presence and orientation of a QR Code in an image

3.12.2 Alignment Markers

Figure 3.26: Alignment Markers

They assist to straighten out QR Codes printed on a curved surface since they are smaller in size than the position detection markers. If a code stores more information, the larger it gets and also requires more orientation patterns.

3.12.3 Timing Pattern Markers

Figure 3.27: Timing Pattern Markers

On the QR Code, alternate black and white elements help to precisely configure the data grid. The scanner determines the size of the data matrix using these lines.

3.12.4 Version Information Markers

Figure 3.28: Version Information Markers

These markers identify which of the 40 different QR Code versions is currently in use. Versions 1 through 7 are the most prevalent.

3.12.5 Format Information Markers

Figure 3.29: Format Information Markers

The format patterns hold information concerning error tolerance and data mask pattern, enabling scanning of the Code easier.

3.12.6 Data and Error Correction Keys

Figure 3.30: Data and Error Correction Keys

All of the data is stored in the error correction mechanism included into the QR Code structure, which also shares space with error correction blocks that authorize up to 30% of the Code to be corrupted.

3.12.7 Quiet Zone

Figure 3.31: Quiet Zone

For the scanning application, the quiet zone is essential for distinguishing the QR Code from its surroundings.

3.13 ArUco Marker: How it Works

Many computer vision applications rely heavily on pose estimation. This method is based on the discovery of correspondences between points in the real world and their projected 2D representation. Because this is normally a challenging task, synthetic or fiducial markers are frequently used to make it easier. The usage of ArUco Markers is one of the most effective methods. The fundamental advantage of these markers is that a single one produces enough speculations (its four corners) to achieve the camera pose. Furthermore, the inner binary codification facilitates error detection and repair techniques to be used. This marker is generally put on the photographed item or scene. It's a binary square with a black background and edges, generating a white pattern that identifies it. The black edge makes it easier to detect them. They're available in a range of sizes. For a successful detection, the size is chosen based on the item size and the scene. If very small markers cannot be detected, expanding their size can assist them to be detected. Using OpenCV, these markers can be easily generated. There are 25 predefined dictionaries of markers in the aruco module of OpenCV. A dictionary's markers all have the same number of blocks or bits (4×4, 5×5, 6×6, or 7×7), and each dictionary has a specific amount of markers (50, 100, 250 or 1000).

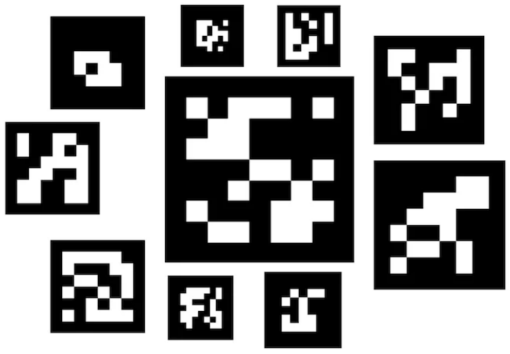
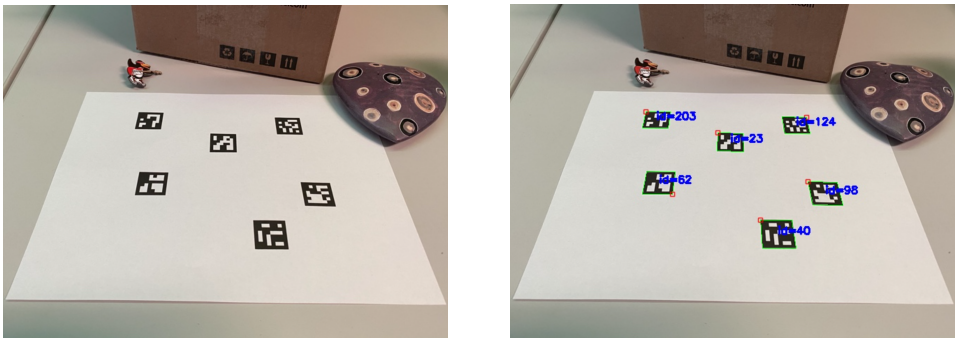


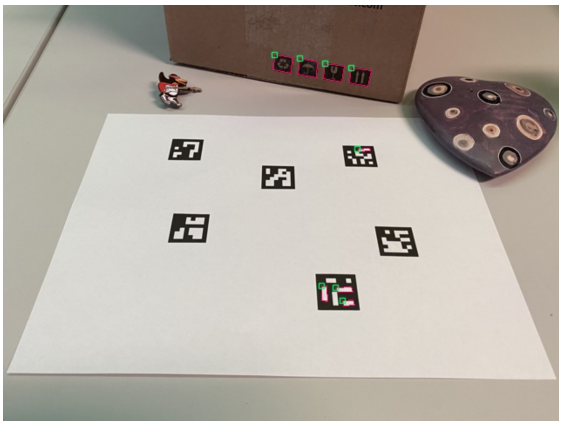
Figure 3.32: ArUco Markers Examples[10]

There are two basic steps in the marker detecting process:

- The first one is candidate marker detection which is rotation independent. The image is evaluated in this step to locate square forms that could be candidates for markers. It starts with adaptive thresholding to separate the markers, then retrieving contours from the thresholded image and discarding those that are not approximate to a square shape.
- After detecting candidates, it's time to see if they're genuine ArUco Markers by looking at their inner codification. This phase begins by removing each marker's marker bits. To accomplish so, the marker is first transformed into its canonical form via a perspective transformation. The canonical image is then preprocessed to distinguish white and black bits using Otsu. The image is split into cells based on the size of the marker and the size of the border. The amount of black or white pixels in each cell is then counted to determine whether the bit is white or black. Lastly, the bits are examined to check if the marker corresponds to the dictionary in question. When required, error correction methods are used.



(a) (b)



(c)

Figure 3.33: (a) Image containing a printout of ArUco Markers. (b) Image in which ArUco markers are detected (in green). Some markers are rotated and the upper left corner of the marker is marked by a little red square. (c) Image in which the candidate markers were discarded during the detection process (in pink).

3.13.1 Overview of the Algorithm

The algorithm [82] generates dictionaries based on a strategy that maximizes the distance between markers and the number of bit transitions. In the process, the maximum theoretical distance between two square binary markers is found. Then, using the created dictionaries, a method for automatically identifying markers in photos and fixing probable errors is proposed. Unlike prior approaches, which impose predefined dictionaries, this system presents an automatic technique for

creating them with the necessary number of markers and bits. The goal is to choose m markers from the domain of all markers with n×n bits, $\mathbb{D}$, that are as far apart as feasible with the fewest bit transitions. The goal is to formulate the dictionary $\mathcal{D}^*$ that optimizes the required criterion $\hat{\tau}$(D)): $\mathfrak{D}^* = \underset{\mathfrak{D}\in\mathbb{D}}{\operatorname{argmax}}\{\hat{f}(\mathcal{D})\}$

The algorithm starts with an empty dictionary $\mathcal{D}$, which is gradually loaded with new markers. The markers are represented as a (n+2) X (n+2) grid, with the external cells set to black, resulting in a visible outer border. The n×n cells that remain are used for coding. Thus, a marker is defined, $m = (w_0, w_1, \ldots, w_{n-1})$ as a dataset composed of n number of binary words w each of which length is n such that, $w = (b_0, \ldots, b_{n-1} \mid b_i \in \{0, 1\})$

$\mathbb{W}$ is denoted as the set of all potential n-bit words, with the cardinal $\mathbb{W} = 2^n$.

Each iteration of the algorithm chooses a marker based on a stochastic process that allots a higher probability to markers with a greater transition distance and whose words have not been included in $\mathbb{D}$ yet. It is added if the distance between the created marker and the ones in $\mathbb{D}$ is greater than a minimal value τ. Otherwise, the created marker is discarded, and a new one is chosen at random. When the desired amount of markers is obtained, the process comes to an end.

3.13.2 Generation of Markers

The proposed method for creating a marker entails choosing n words from $\mathbb{W}$ and replacing them. To accomplish this, the probability of each word $w_i \in \mathbb{W}$ being chosen at each iteration is described as, $P\{w = w_i\} = \frac{T(w_i)O(w_i,\mathcal{D})}{\sum w_j \in w T(w_j)O(w_j,\mathcal{D})}$

The probability of selecting a word is defined as a combination of two functions in the above equation. The first, $T(w_i) \in [0, 1]$, refers to the number of bit transitions in the word. As the number of transitions between subsequent bits grows, $T(w_i)$ tends to 1; as the number of transitions drops, $T(w_i)$ tends to 0. The function $O(w_i, \mathcal{D})$ counts the amount of times the word W_i shows up among the markers in $\mathbb{D}$. The goal is to limit the probability of selecting words which have already been chosen numerous times.

3.13.3 Calculation of Distance

The distance between two markers is defined as, $D\left(m_i, m_j\right) = \min_{k\in[0,1,2,3]}\left\{H\left(m_i, R_k\left(m_j\right)\right)\right\}$

The Hamming distance between two markers is defined as the total of Hamming distances between each pair of marker words and is represented by the function H. The R_k function rotates the marker grid $k \times 90°$ in a clockwise direction. Between the markers, the rotation-invariant Hamming distance is thus defined by the function D. The distance between a marker and a dictionary is measured as follows: $D\left(m_i, \Sigma\right) = \min_{m_j\in\mathcal{D}}\left\{D\left(m_i, m_j\right)\right\}$

The above equation is the distance of the marker to the nearest one in the dictionary. Finally, not only the markers are differentiated from one another, but the orientation of the markers are also recognized in this method. Pose estimation would be unsuccessful otherwise. Therefore, this method addresses a marker as valid when it ensures that the minimum distance between its own rotations is more than . The marker self-distance of this method is defined as: $\mathcal{S}\left(m_i\right) = \min_{k\in(1,2,3)}\left\{H\left(m_i, R_k\left(m_i\right)\right)\right\}$

In conclusion, if both $\mathcal{S}\left(m_i\right)$ and $D\left(m_i, \mathcal{D}\right)$ are more than or equal to , this method inserts a marker to the dictionary. If not, the marker is discarded and a new one is created.

3.13.4 Marker Code Detection and Error Correction

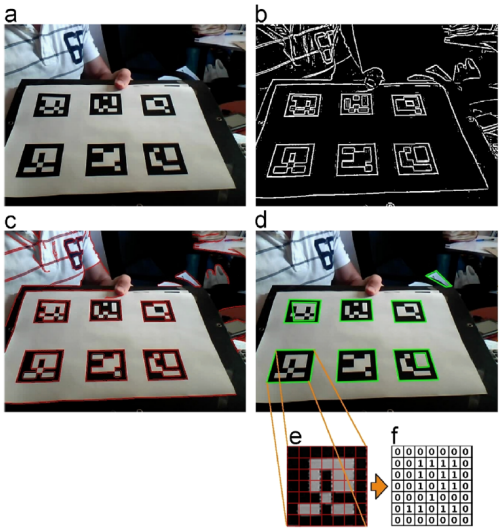


Figure 3.34: (a) Original image containing a printout of ArUco Markers. (b) Output after application of local thresholding. (c) Detection of contours from the image. (d) Approximation of polygonal and discarding of irrelevant contours. (e) Marker example after transformation of perspective. (f) Assignment of bits for each cell of the marker example.

Although the image analysis of this method is not novel, the marker code detection and error correction is a new methodology created exclusively for the generated dictionaries. The following stages are implemented by the system:

- The first step extracts the most visible contours in the grayscale images. They chose a local adaptive thresholding strategy in this study, which has proven to be particularly resistant to varying lighting settings (Figure 3.14(b)).
- After that, the Suzuki and Abe algorithm [83] is used to extract contours from the thresholded image generating a collection of image contours (Figure 3.14(c)), the majority of which are irrelevant. The Douglas–Peucker algorithm [84] is then used to perform a polygonal approximation. Markers that are not close to 4-vertex polygons are eliminated since they are enclosed in rectangular contours. Finally, only the external contours remain after the near contours have been simplified. The produced polygons from this method are shown in Figure 3.14(d).

- The next step is to examine the interior region of these contours in order to determine their internal code. The homography matrix is used to remove perspective projection (Figure 3.14(e)). Otsu's approach [85] is used to determine the ideal threshold value for the image, given that the distribution of images is bimodal. For each pixel that falls within the grid, it is assigned the number 0 or 1 depending on how many pixels fall within that grid (Figure 3.14(e) and Figure 3.14(f)). The existence of the black border is one of the first rejection tests. The inner grid is evaluated using the method described in the following section, if all bits on the border are zero.

- At this stage, it is crucial to establish which of the marker candidates generated truly belongs to the dictionary and which simply exists in the environment. Four separate identifiers are obtained after the code of a marker candidate is retrieved. The candidate is considered to be a valid marker if any of them are found in $\mathbb{D}$. The dictionary items are arranged as a balanced binary tree to accelerate the procedure. Markers are denoted in this way by the integer value generated by concatenating all of their bits. The authors determined that this procedure has a logarithmic complexity of $O\left(4\log_2(|\mathfrak{D}|)\right)$, where the factor 4 implies that every rotation of the marker candidate requires one search.

If there is no match, the correction procedure is used. Given that the smallest distance between each pair of markers in $\mathbb{D}$ is $\hat{\tau}$, a maximum of $\lfloor(\hat{\tau}-1)/2\rfloor$ bits of error can be detected and fixed. As a result, the marker correction approach entails calculating the distance between the erroneous candidate and all of the markers in $\mathbb{D}$. The approach considers the nearest marker to be the correct one if the distance is equal to or less than $\lfloor(\hat{\tau}-1)/2\rfloor$. However, because each rotation of the candidate is compared to the whole dictionary, this method has a linear complexity of $O(4|\mathfrak{D}|)$. In contrast to ARToolKitPlus's dictionaries, which are unable to correct errors, and ARTag's dictionaries, which can only correct two bit errors, this method can repair errors of $\lfloor(\hat{\tau}-1)/2\rfloor$ bits. As a result, this method can fix errors of the dictionary upto 5 bits. It can also produce markers with more bits, resulting in a larger $\hat{\tau}$, hence enhancing the correcting abilities. This detection and correction process is essentially a comprehensive model that can be

used with any dictionary, including the ARToolKitPlus and ARTag dictionaries. In fact, if this approach is used with the ARTag dictionary of 30 markers, it can correct 5 bit errors rather than the only 2 bit error that the ARTag dictionary can correct.

CHAPTER 4

METHODOLOGY

In this chapter, we discuss how we implemented our proposed system. All implementations were done using Python 3.9.6 and its various modules and frameworks. Here we discuss our step by step procedures we followed in implementing the project.

4.1 Dataset Description

We took a more data-driven approach towards the classification of supermarket products. The dataset was created in different phases, and we will be comparing the models trained on different versions of the dataset later on in the report. Here we will discuss in detail the various versions of the datasets we built.

Primarily, we listed 30 of the most commonly found supermarket items. 25 of which are packaged products and the other 5 are weight dependent grocery items. The list of products we chose for our dataset are as follows:

1. ACI Pure Salt 1kg
2. Parachute Coconut Oil 50ml
3. Energy Biscuit
4. Dettol Original Soap

5. Mr. Twist
6. Teer Sugar 1kg
7. Rupchanda Cooking Oil 1L
8. Pran Hot Sauce
9. Cocola Chicken Masala Noodles
10. Frutika Grape 250ml
11. Sepnil HandSanitizer 40ml
12. Snickers
13. Sprite 250ml Bottle
14. Vaseline Original
15. Coca-cola 250ml bottle
16. Lux Soft Glow
17. Dettol Soap Aloe Vera
18. Neem Soap Pure Neem
19. Neem Soap Olive and Aloe Vera
20. Pears Soap with Lemon Flower Extracts
21. Pears Soap with Natural Oils
22. Maggi Soup Chicken
23. Maggi Soup Vegetable
24. Fresh Coriander Powder
25. Fresh Turmeric Powder
26. Miniket Rice (Weight Dependent)
27. Local Beef (Weight Dependent)

28. Local Onion (Weight Dependent)

29. Local Potato (Weight Dependent)

30. Local Garlic (Weight Dependent)

Figure 4.1: Products in the dataset

We created the first version of the dataset was created with mostly single images of the products. Then we added synthetic images to enhance our dataset even further and finally, we added some stacked images to the datasets in order to improve detection performance.

Table 4.1: Versions of datasets

Dataset Version	Description
Version-1	Mostly single product images
Version-2	Synthetic Data was added to the dataset
Version-3	Synthetic Data was removed from the validation set
Version-4	55 Images were added with highly dense product arrangement
Version-5	14 new products were added including weight dependent ones.

Throughout this chapter when we mention our dataset, we shall be mostly referring to Version-5. We shall discuss further the rest of the datasets in later chapters.

4.1.1 Visually Similar Looking Products

On the Version-5 of our dataset we have added 10 visually similar looking products, most of which are different variants of the same product. The side by side comparison of similar pairs are shown for both their back and front labels. An idea of what they look like may help better interpret our results and analyses in the upcoming chapters.

Figure 4.2: Visually Similar Product Comparison

4.1.2 Data Acquisition

4.1.2.1 Daraz Review Image Scrapping

A comprehensive dataset consisting of our local retail products is rare to find. So we created our own dataset of retail products. In our dataset, we mostly accumulated data from 2 sources. Firstly, we have found that the most diverse images for a single product can be found in the review section of Daraz [86]. Daraz is an e-commerce site where people buy local goods. The review section of each product contains multiple images of the products they receive upon delivery.

We planned to capitalize on these images by creating a web-scraper that will iterate through each product review, download the review images, and collect them in a single folder. We made the web scraper in Python 3.9 using the modules Selenium 4.1.0 [87] and BeautifulSoup4 4.10.0 [88].

Beautiful Soup is an HTML Parser. It has tools to retrieve the HTML data of a webpage. It requests the data from the webserver and stores them. The data is referred to as soup. However, there is a limitation that it can only retrieve the information a webpage displays on load. If some elements are loaded later via JavaScript may not be represented in the HTML data.

In order to get the review images to load before parsing HTML data, we needed to scroll down to the end of the reviews and keep moving to the following review pages. We achieved this by using Selenium which is a web driver. It drives a browser of choice as a human user would. It is possible to click buttons, scroll and perform all the interactions with a website like a human user.

We have used Selenium to drive Google Chrome on to load the product page and scroll down to the end of the current review section. After all the review images are loaded, we use beautiful soup to parse the HTML data and extract links to the images from that data. After that, we iteratively download the images from those links. Then we scroll up and press the next button in the review section and scroll down again to load all the photos and keep following the same procedure until we can't find any more product images.

4.1.2.2 Manual Photography

We couldn't find all the products or enough images of some products from Daraz. So we manually photographed some of the products. The photos were taken using a smartphone camera against various backgrounds and lighting conditions. They were photographed both as individuals and as groups stacked together. We kept partial overlapping of products in some images but avoided any significant overlap for our model to properly recognize the product labels.

4.1.3 Data Cleaning

After acquiring the data, we had to clean the data, removing duplicates and unusable images. We had to deal with the fact that people didn't always upload exact pictures of the product in the review sections and some photos just focused on the expiry date rather than the entire product label. There were a lot of duplicate images as well.

We automated our duplicate image detection by using dupeGuru. It is a Python-based cross-platform tool for finding duplicate files. We provided a link to each product folder and the application automatically detected the duplicate files and removed them [89].

We have manually searched through the downloaded and photographed images for blurred images, images with products not correctly visible, images with significant overlapping, etc., and removed them from our dataset.

4.1.4 Pre-Processing

For us to train our model on the data, we had to first process the images. YOLO works great on square images. There is also a trade-off between training speed and features when choosing the resolution of the images. If we choose a large resolution, it will be easier for the model to pick up more features while the training time will increase significantly. On the other hand, choosing too small a resolution will compress the features and lead to poor classification. So we decided to use 416x416

resolution for our model. But most of our images were not square. So we had to resize them. But if we wanted to crop them, we had to manually go through each image as the products were located in various parts of the images, and blindly cropping them could lead to data loss or the products being undetectable.

Thus, we resized the images by making them square first by padding the extra parts with 0s and creating a black region at the sides of the images. This was done using PIL, Matplotlib, and OpenCV. We wrote a script to transform all the images into 416x416 resolution JPG format.

4.1.5 Data Annotation

While there are many different ways for data annotation, we chose to annotate our data manually as in this way we can provide accurately labeled data to our model. So, we have used an annotation tool called Labelimg [90]. It's a graphical interface made using Python and PyQt4 for annotating image data in different formats. It also supports the YOLO format.

We simply had to put our list of classes into its data folder and using its interface, we have drawn bounding boxes around products in each image and selected their class. Then it automatically generates the text file in the format mentioned above.

Figure 4.3: Correct vs Incorrect Annotation

It's worth mentioning that we have tried our best to make an exact bounding box around the object, not including extra areas as it can hamper the model's accuracy. One of the most crucial accuracy metrics for an object detection model is IoU or Intersection over the union. If we have extra spaces in our bounding box, this metric decreases.

4.1.6 Synthetic Data

We chose to augment our dataset with synthetic data in order to enhance the overall performance of our model. In this section we have discussed the techniques for synthetic data generation we have used.

In order to create synthetic images, we extracted transparent PNG images of each object (both front and backside) using Adobe Photoshop. Then we collected around 20 different surface textures and placed products randomly on each of the background images. The position, rotation, and scaling of the products were randomized. Each synthetic image had 1-10 products in them which in turn lead the number of synthetic instances to far surpass the number of authentic instances in the dataset.

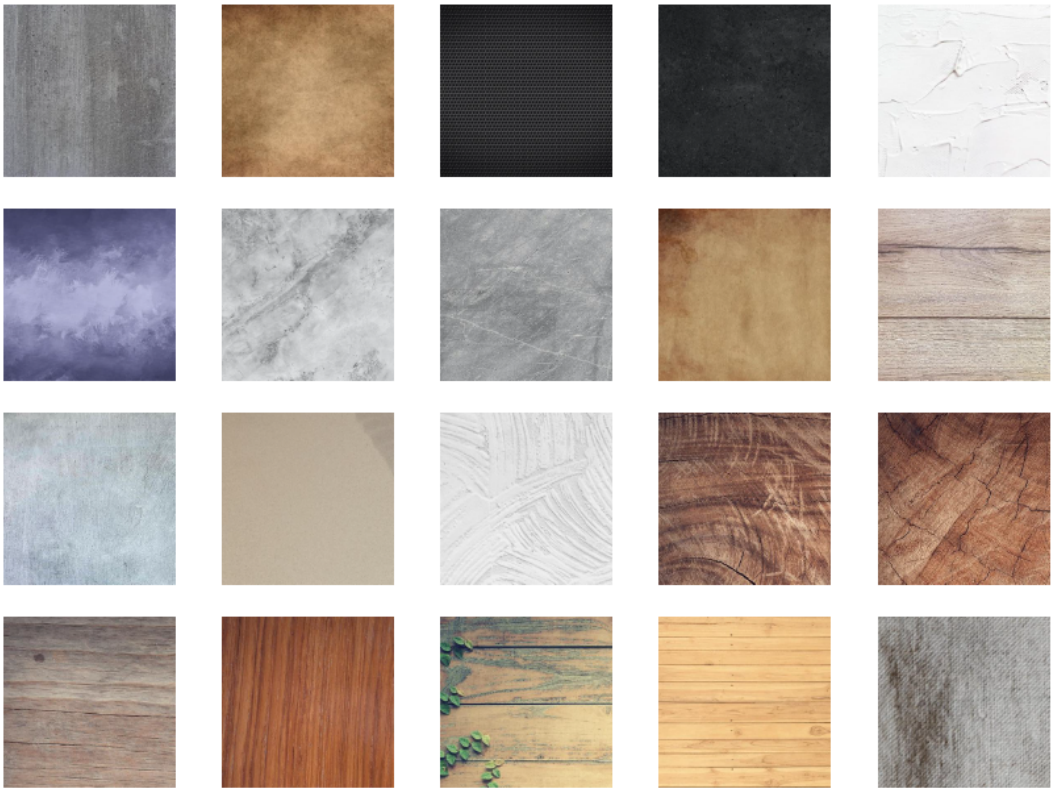


Figure 4.4: Surface textures for synthetic image generation

It requires the background and object files to be in separate folders. Each object category needs to be in its subfolders. The script then randomly chooses some number of objects and loads the object PNG files and randomly translates, rotates, and scales them across a randomly selected background image. It creates the specified number of synthetic images in the same way.

Figure 4.5: Synthetic Image

This placement of images was done using the Flip Github repository. It takes the PNG images of the objects and the background images as input and generates any specified number of random Synthetic images. There is some discrepancy in the scale of the images relative to each other as no relative size ratio is fixed provided in the algorithm. But this doesn't hurt the accuracy of the model very much, rather the inclusion of different sizes of the products helps it to be able to better recognize the product from different distances. We initially have created 150 synthetic images per class.

4.1.6.1 Annotating Synthetic Data

The annotations for the bounding boxes in the Synthetic images were automatically created on generation. Flip used Pascal VOC format for annotation so we have to convert the format to YOLO text files before using them [91]

Pascal VOC format uses XML files to store annotation details. There are extra parameters in this format than what YOLO needs like name, truncated, pose, difficult, etc also the bounding box format for Pascal VOC is different (x_min, y_min, x_max, y_max). So, in order to convert the annotation to YOLO format, we must convert the bounding box to Yolo format and replace the object name with its id number. [92]

$$norm\_x\_center = \frac{(x\_max + x\_min)}{2 * img\_width}$$

$$norm\_y\_center = \frac{(y\_max + y\_min)}{2 * img\_height}$$

$$norm\_width = \frac{(x\_max - x\_min)}{img\_width}$$

$$norm\_height = \frac{(x\_max - x\_min)}{img\_height}$$

We created a python script for automatically converting the Pascal VOC XML files to YOLO Format text files. After running the conversion the synthetic images have been ready to be added to the dataset

4.1.7 Dataset Distribution

The dataset was initially split between with an 80, 10, 10 ratio between the train, validation, and test slices. Then the synthetic data was added in the aforementioned process. We have 13383 object instances spanning across 5734 images. The instance distribution among the 3 slices for the final version of the dataset are shown on Figure 4.6 and 4.7. And all synthetic values for the training dataset have been placed only in the training slice. We have a total of 3100 synthetic images in the dataset. The distribution of authentic and synthetic data in the training slice is illustrated on Figure 4.8 and 4.9.

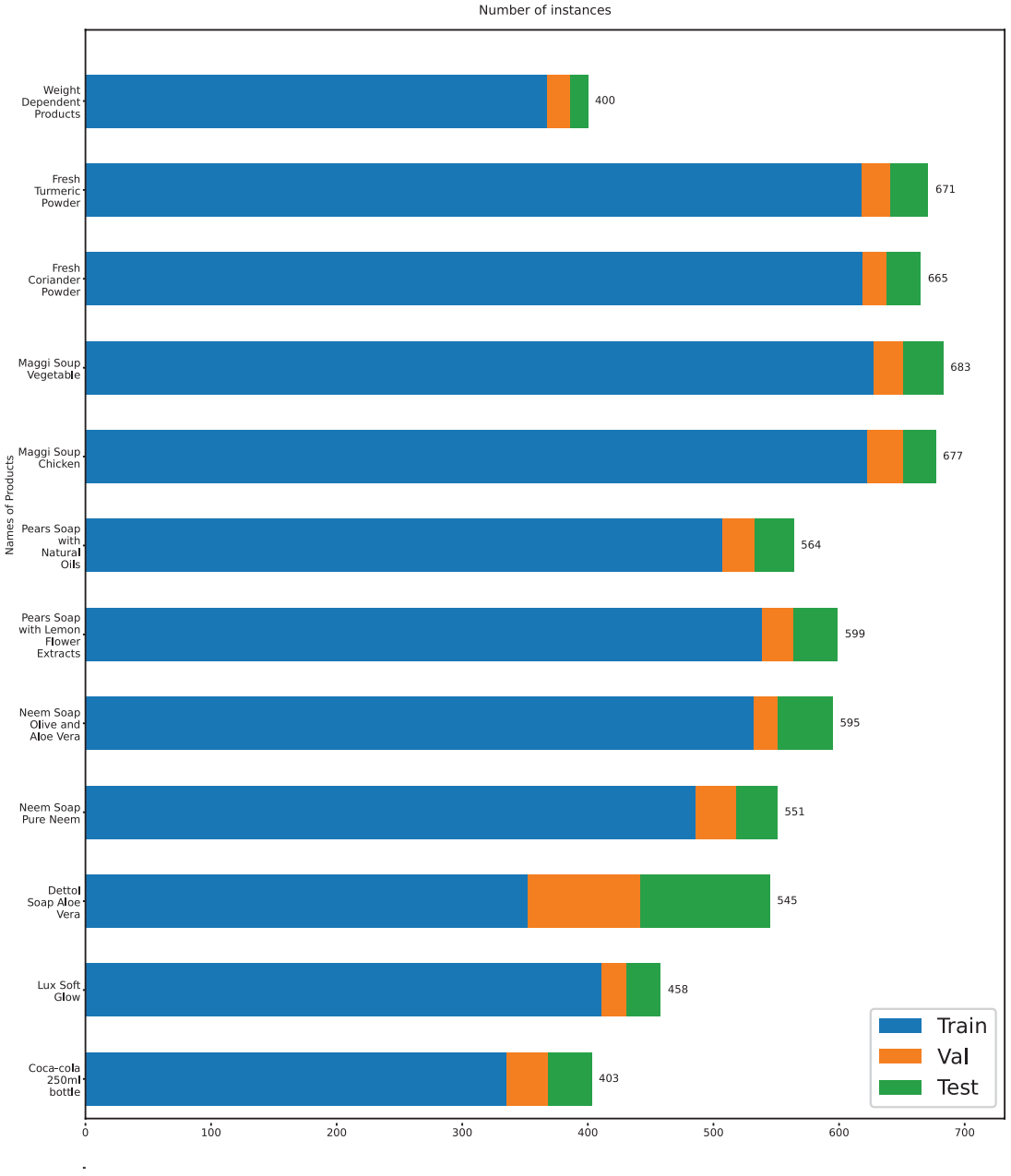


Figure 4.6: Distribution of products in dataset splits

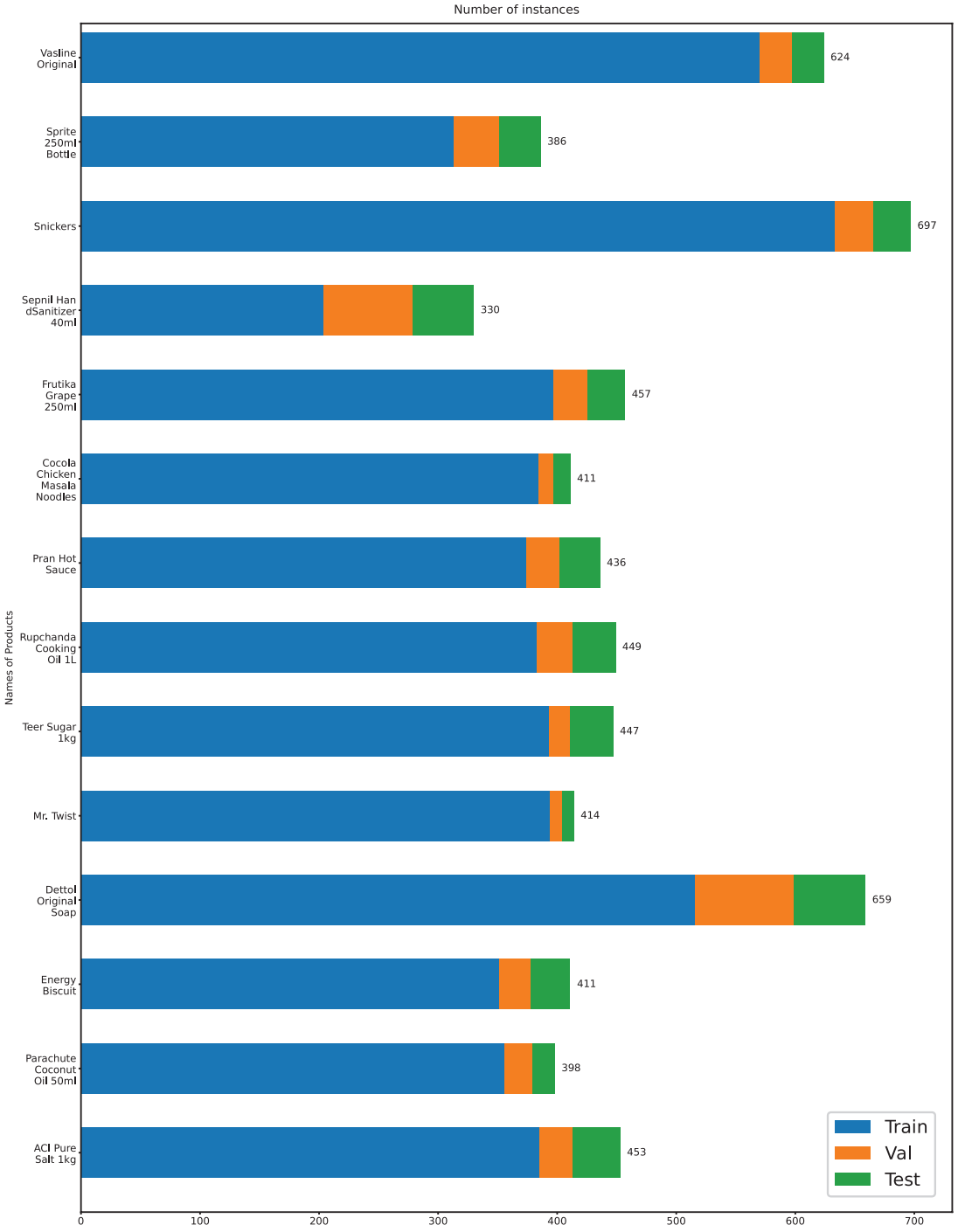


Figure 4.7: Distribution of products in dataset splits

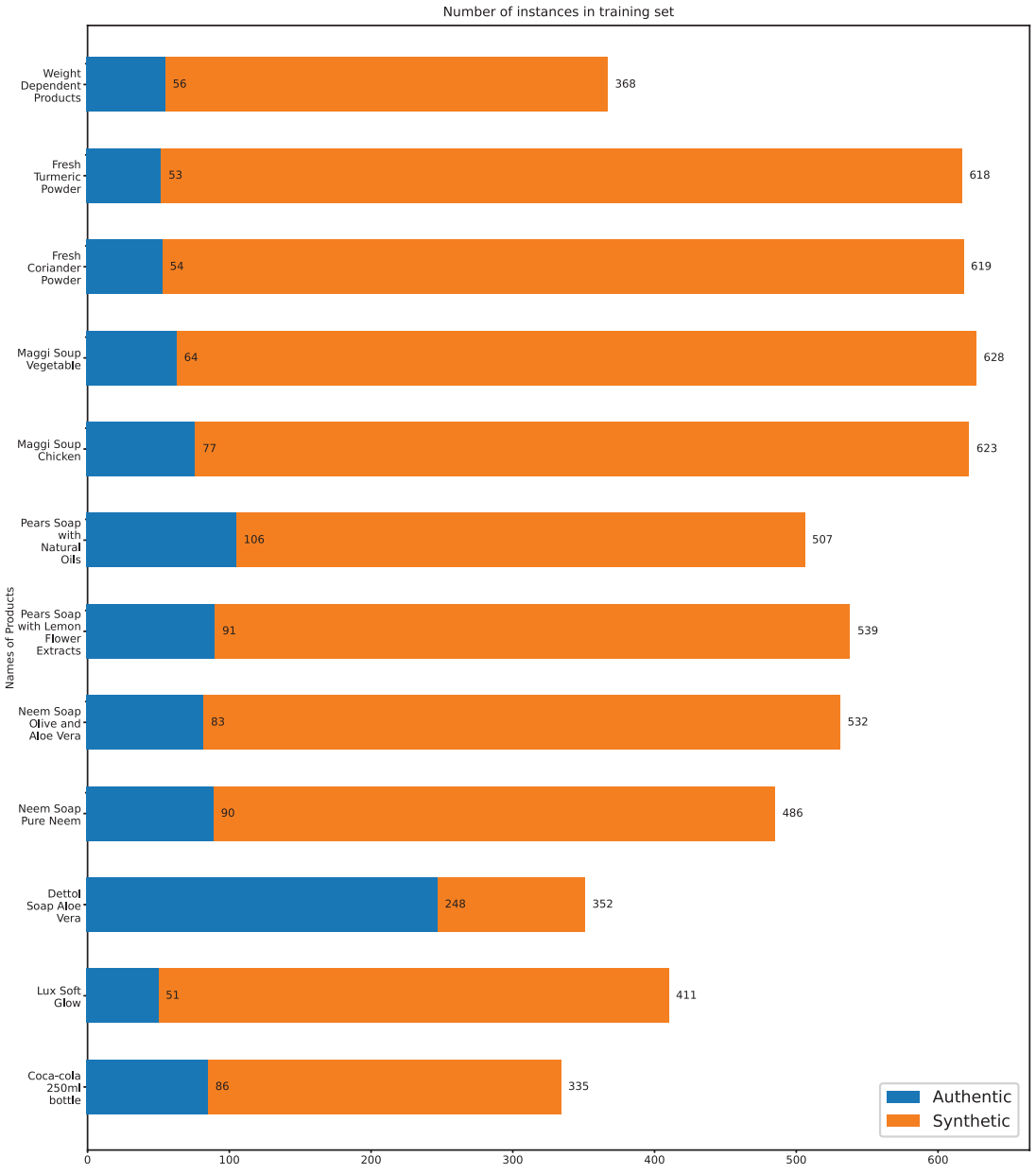


Figure 4.8: Distribution of Synthetic and Authentic images in training set

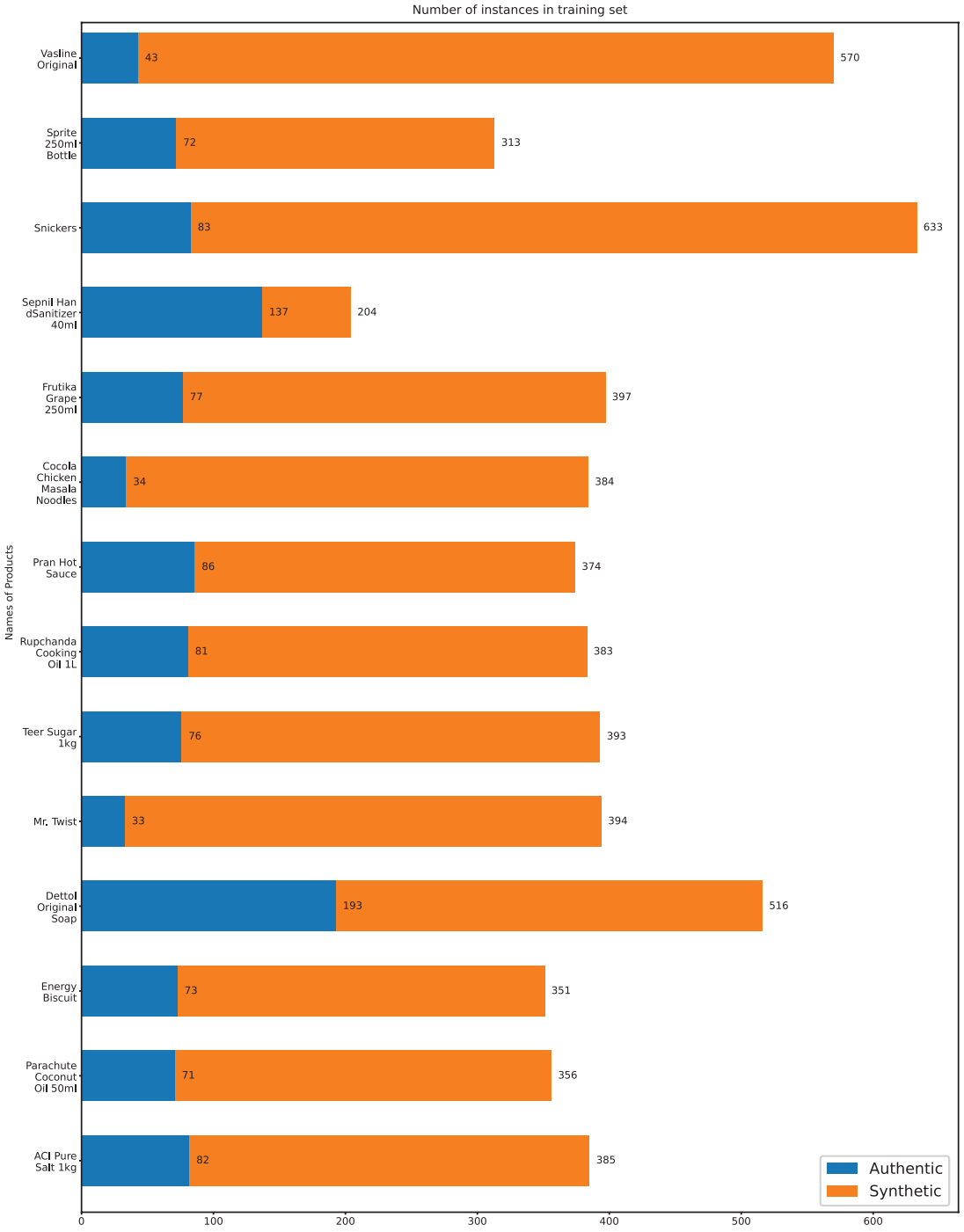


Figure 4.9: Distribution of Synthetic and Authentic images in training set

4.2 Model Overview

The algorithm YOLOv5 has been implemented with Pytorch. It's a machine learning platform that emphasizes both the aspect of speed and usability. Pytorch is based on Python, so it is compatible with all the renowned libraries for scientific computation. It also supports GPU acceleration for maximum performance for maximum computing power. [93]

Yolov5 has multiple variations of models depending on size and complexity. There are mainly 4 variations:

1. Yolov5s
2. Yolov5m
3. Yolov5l
4. Yolov5x

We took a transfer learning approach by loading pre-trained weights that were trained up to 300 epochs on the COCO dataset. Then trained the models on our dataset starting from the pre-trained weights as a checkpoint and trained up to 100 epochs. Then saved the best weights that we obtained from the training.

COCO or Common Objects in Context is a dataset by Mircosoft which features a wide range of common object instances. The dataset is designed for object recognition with an understanding of the entire scene. There are 2.5 million instances of common everyday objects spanning across 328,000 images that are labelled for bounding boxes and object instance segmentation[94].

The pretraining information on the COCO dataset for the YOLO models are showcased below [95]:

Table 4.2: Pretraining information on the COCO dataset

Model	Size (pxl)	mAPval 0.5:0.95	mAPtest 0.5:0.95	mAPval 0.5	SpeedV100 (ms)	FLOPs 640(B)
s	640	36.7	36.7	55.4	2.0	17.0
m	640	44.5	44.5	63.1	2.7	51.3
l	640	48.2	48.2	66.9	3.8	115.4
x	640	50.4	50.4	68.8	6.1	218.8

We only trained the Yolov5s, Yolov5m and Yolov5l models due to GPU resource limitations. We can infer from the graph below that larger models need more GPU processing power. We figured that the optimum point for our available hardware would be Yolov5m or medium model.

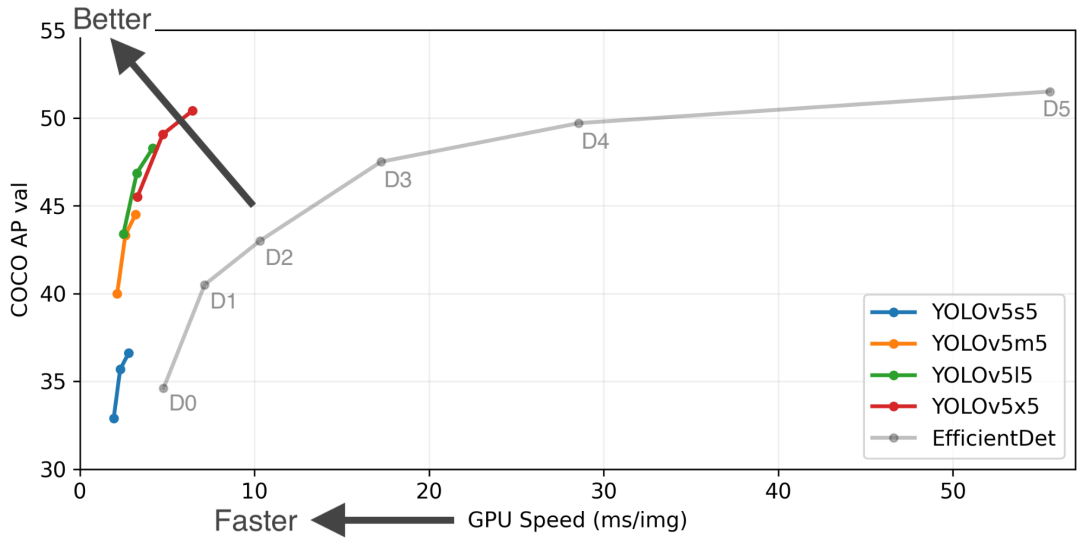


Figure 4.10: Comparison of Yolov5 Models

The difference between the small, medium and large models is that the small one has layer channel multiple and model depth values of 0.5 and 0.33 compared to 0.75 and 0.67 of the medium model, 1.0 and 1.0 of the large model. They have similar overall structures. The structures for the three models are show on Figure 4.11

Figure 4.11: Yolov5 Architecture [11]

4.2.1 Hyperparameter Evolution

We have used a Genetic algorithm-based hyper-parameter evolution technique to tune our model further. Each generation to 10 epochs starting from the default hyperparameter, which is optimized on the COCO dataset.

The fitness function is defined as:

$$F = 0.1 * (mAp_{0.5}) + 0.9 * (mAp_{0.5-0.95})$$

This algorithm has an 80% probability of mutation with a 20% variance. Instances are created until the mutations occur. Then the best mutation of each generation is used to create the next generation in the same way. At last, the hyperparameters for the best generation are saved.

We trained a total of 150 generations and ended up with the best hyperparameter settings on the 11th generation. Even though it was recommended to run 300 generations, we stopped early due to lack of GPU power.

4.2.2 Training Description

For our final model we created an ensemble with Yolov5m and Yolov5l model. Ensemble is a learning paradigm in which a collection of neural networks is trained over the same task[96].Hansen and Salamon [97] proposed this approach, which demonstrates that ensembling a bunch of neural networks, i.e. training several neural networks and then integrating their predictions, can considerably improve a neural network system's generalization ability.

It took around 1 min per epoch ending in around 6hr of total training time. We initialized our models with weights trained 300 epochs on the COCO dataset and used it as a starting checkpoint. Then we trained our Yolov5l model for 100 epochs and Yolov5m model for 300 epochs on our custom dataset that we mentioned above. Then we made an ensemble using these two models. The training description for our main models are listed below:

Hardware:

Table 4.3: Hardware used to train the models

CPU	Intel i7-8700
RAM	32 GB
GPU	MSI GTX 3080 Trio

Settings:

Table 4.4: Training settings

Epoch	300(Yolov5m), 100(Yolov5l)
Batch Size	32
Image Size	416x416
Workers	1
Optimizer	SGD

4.3 Weight Dependent Product Detection

Not all the products in a retail store are pre-packaged. There are products that need to be weighed first before they are sold. For example, rice, meat, fish, vegetables, etc. The weights of these products are sometimes not discrete. The weight of a piece of meat, fish, and vegetable can vary with their size and there are scenarios when a fish or a particular cut of meat is sold as an individual item. In those cases, it is important to bill the customer based on the weight of the product.

For a Computer-vision based system, it is virtually impossible to determine the weight of an object precisely from its image. Therefore, we needed to employ some other mechanism in tandem with object detection. We decided to use some sort of coded marker that will contain its weight and the identity of the product and will be recognizable with a camera. We experimented with two approaches, one based on QR code and another based on a custom marker based on ArUco markers.

4.3.1 Our Approach

We packaged the weight-dependent goods inside identical red bags. We trained our model to detect and classify the red bags as a separate class of products. Each bag was then labeled with a marker corresponding to its contents and weight. The markers can be printed by an automated system at the product-specific sections of the store. During billing, when the bag is detected by our model, we crop out its bounding box and send the cropped image to the marker decoder. The marker decoder then decodes the contents of the marker. Then the final classification is

made based on that. If a product is not weight dependent, it is detected in a single stage. But if it is weight dependent then it is detected in two stages, first as a general category, then the marker is decoded to specify the product along with its weight.

Figure 4.12: Weight Dependent Product Packaging

The product ids are searched and matched from a pre-defined database. For our prototype, we are using a local .csv file but in practical product level applications, an online database will be used in its place.

A block diagram of the complete algorithm is given below:

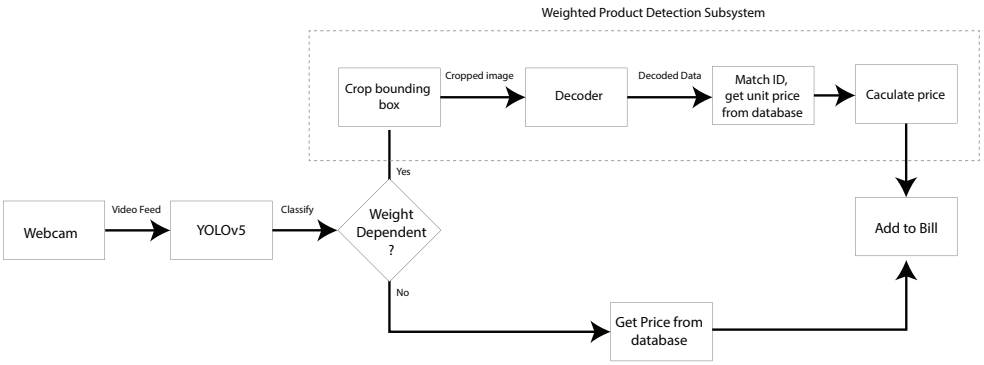


Figure 4.13: Block Diagram of Billing Algorithm

4.3.1.1 QR-Code Marker

For each of the 5 weight-dependent products we generated a QR code. We discussed in detail in the previous chapter, how QR codes can be detected using a camera. The data in our QR codes are stored in the following format: product-id, weight-in-kg.

The QR codes generated for the items are given below:

Figure 4.14: QR codes for products in the dataset

Each QR code was printed with a size of 3.5cm x 3.5cm. We implemented the QR code decoder using the pyzbar library [98]. But as QR codes are designed to store a large number of characters, it is sometimes very complex. And so, if the images get a bit blurry due to camera quality, or distance the markers can no longer be decoded. We have shown our results with QR codes in the next chapter. As lowering the camera too close to the products will decrease FOV, we used a different marker to implement our system.

4.3.1.2 Hybrid ArUco Marker

ArUco markers are simple, orientation-independent and so can be detected very fast even when they are very far away. But the downside of these markers is that they can only represent an integer id up to 1000, which makes sense as they were mainly designed to be used in augmented reality-based applications[99]. We have discussed in detail these markers in the previous chapter. In our case, we need more data space to store both product id and weight. Thus we solved this problem by combining 2 classes of ArUco markers as a single unit.

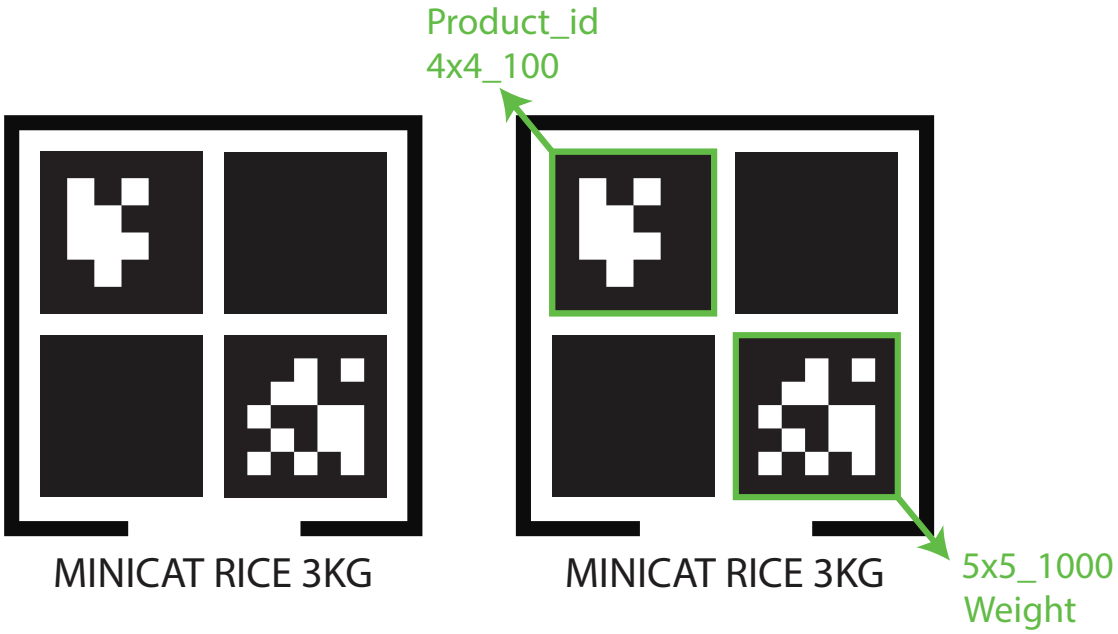


Figure 4.15: Hybrid ArUco Marker

The 4x4_100 marker is used to represent product ID. It is capable of representing 100 different values, so 100 different products can be represented with it, which is more than enough for our prototype. The marker is also scalable as the two black boxes can be used to store additional markers to enhance storage further. The 5x5_1000 marker is used to store the weight of the product as a 3 digit number. In our current prototype, if we divide the number by 100, we get the weight in kilograms. Thus we can store weight value in kilograms up to 2 decimal spaces which is sufficient for groceries.

Decoding: The hybrid ArUco marker can be decoded very simply by just using a normal ArUco marker decoder in two stages. At first, the image has to be scanned for markers corresponding to the ArUco 4x4_100 dictionary, then that value is stored and the image is again scanned for markers corresponding to the ArUco 5x5_1000 dictionary. We have implemented this detection using the OpenCV library which has built-in functions for basic ArUco marker detection. The markers generated for the items are given below:

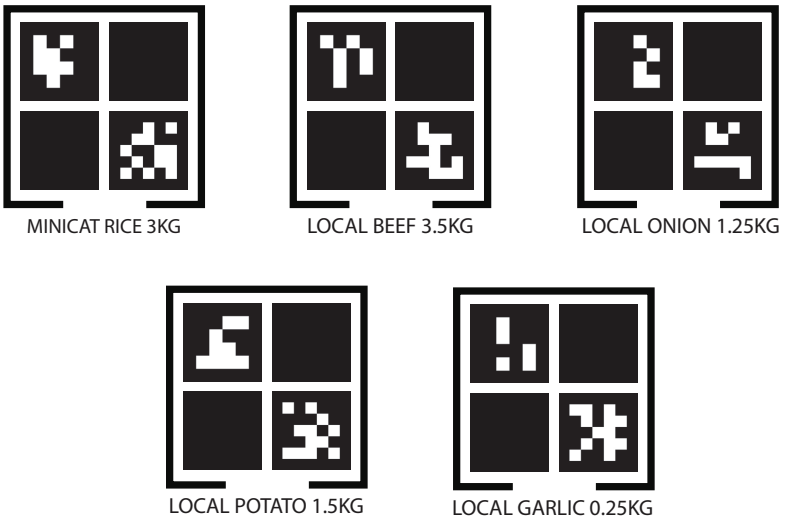


Figure 4.16: QR codes for products in the dataset

Figure 4.17: Hybrid ArUco Marker Detection

The printed version of these markers are of the same size as the QR codes but they demonstrate much reliable detection. We have shown a comparison of performance between the Hybrid ArUco Marker and QR code in the next chapter.

4.4 Hardware Setup

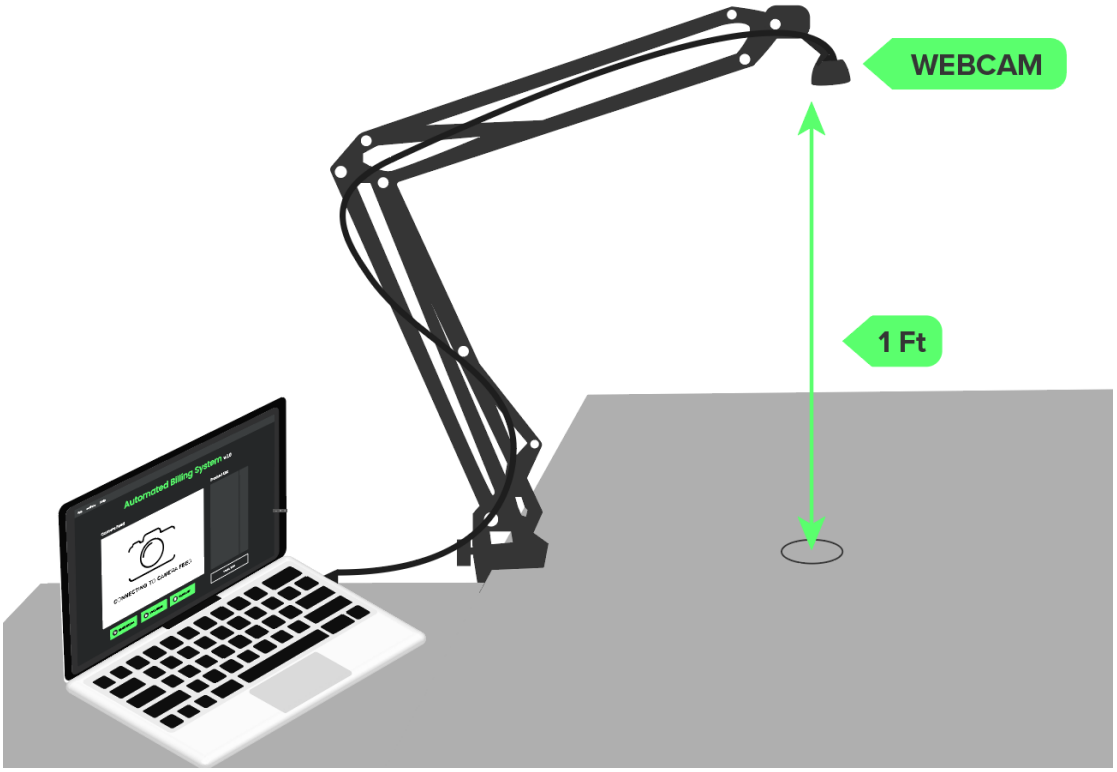


Figure 4.18: Hardware Setup

We constructed a table-mounted setup with our components. The setup consists of the following:

1. Webcam
2. Adjustable Table mounted arm
3. Computer
4. Table

The webcam is kept approximately 1 foot above the surface of the Table using the adjustable table-mounted arm. The surface of the Table should have plain colours devoid of complex patterns to aid with the detection. The webcam is connected to the computer via a USB cable, as shown in the figure above. Products will be placed in batches under the webcam for detection and billing.

4.5 GUI Implementation

GUI or Graphical User Interface is an interaction mechanism where a user can use a pointing device (perhaps a mouse) to interact with different functionalities of a program. This brought about a revolutionary change in the field of Human-Computer Interaction (HCI) in modern times. We planned to implement a GUI base to provide the cashier with an easy-to-use application for our billing system [100].

4.5.1 Objectives of the Application

The list of features we have considered implementing are as follows:

1. Initiating and terminating a billing session
2. Clearing existing list
3. Generating a printable receipt
4. Dynamically updating prices to a database
5. Automatic refreshing of the product list when products are changed
6. Mechanism to lock an existing batch of products on the list

Based on the features, we have first created a wireframe of the GUI layout.

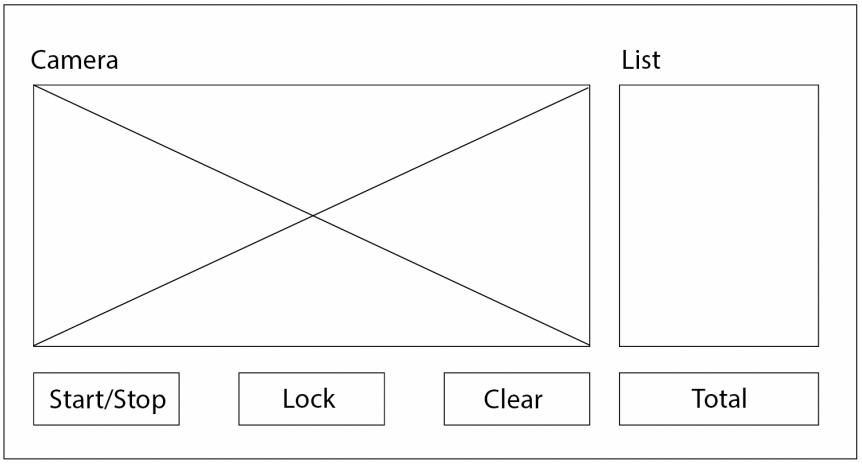


Figure 4.19: Wireframe of the app

4.5.2 Description of the UI

The UI design was done based on the wireframe diagram, using Adobe Illustrator. Then the GUI was implemented using the PyQt5 framework [101].

Qt is a cross-platform C++-based programming toolkit that offers everything to build an app UI. PyQt is the collection of bindings that glue together Qt and the vast array of modules that we get access to with Python.

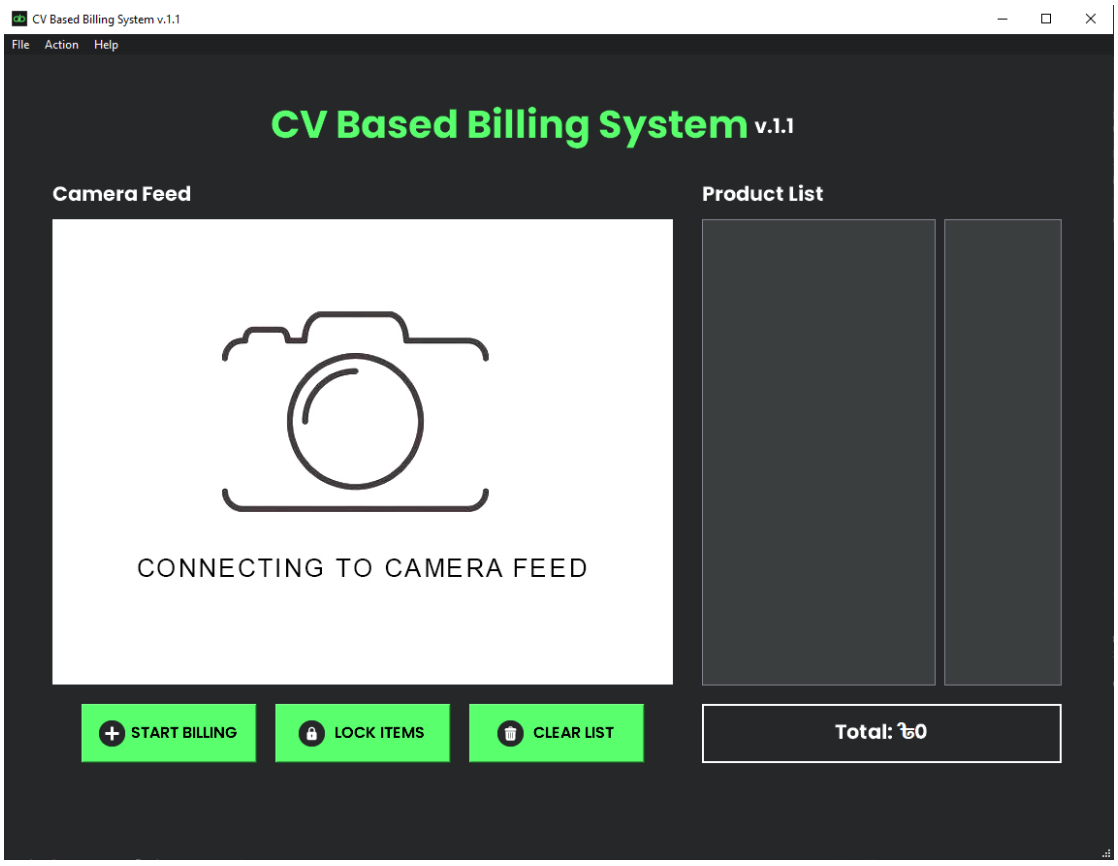


Figure 4.20: GUI of the Application

Camera Feed: In the app, the Camera Feed window displays the camera view in real-time. Any products that are detected are marked with a bounding box around them, and their name is displayed.

Product List: When the billing starts, the detected products are listed here. The first column shows the name and amount of the product in this format: n x Product_name. And the second column lists the value of those products in Taka.

Buttons:

- The "Start Billing" button initiates the billing process. The application won't list the products before the button is pressed. It's a toggle button, so it changes to "Stop billing" after billing starts. To stop the billing process, we have to press the same button again.

- The “Lock Items” button is for locking down the batch of products currently on the list.
- The “Clear List” button clears all the current entries in the list.
- The “Stop Billing” button terminates the billing process and generates a receipt in a text file.

All of the functionalities are tied to keyboard shortcuts for ease of use of the cashier.

Table 4.5: Shortcuts for the App

Functionality	Keyboard Shortcut
Start billing	Space
Lock items	S
Clear list	X
Stop billing	Space

4.5.3 Highlight Features of the App

4.5.3.1 Automatic Refresh

Usually, no object detection system is 100% accurate under every scenario. Thus, the application may momentarily miss-classify the products when the cashier places them on the checkout area, or one product is significantly overlapped by the other.

The application automatically updates the list on each frame to tackle this issue. So that if a product is miss-classified on one frame, the application will dump the data and refresh it on the next frame automatically, or the cashier can slightly rearrange the products on demand to aid it classifies the products correctly.

4.5.3.2 Batch Locking Mechanism

One of our main objectives is to make the process of supermarket checkout faster by enabling the cashier a way to bill multiple products at once rather than one at a time. If we could bill all the products together, then it would have been most efficient. But there is no practical way to achieve that as cameras have a limited Field of View (FOV). We circumvent this limitation by introducing a way to bill the products in batches.

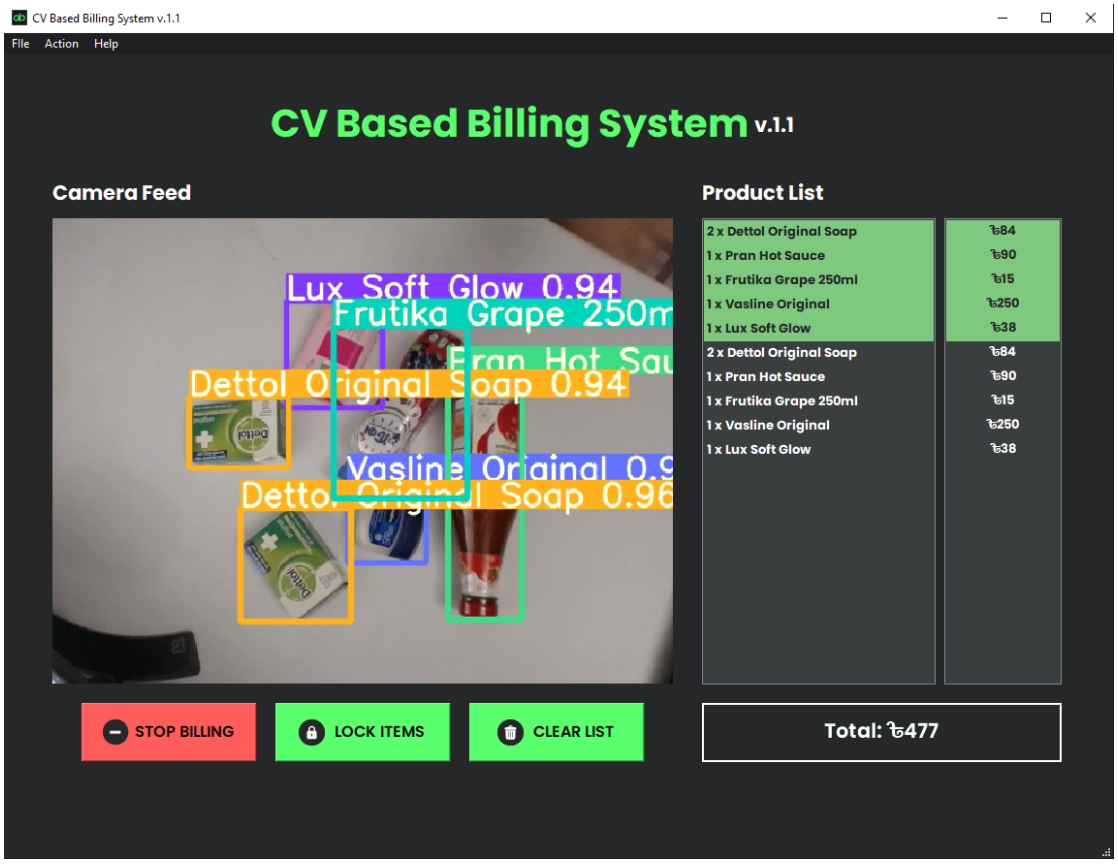


Figure 4.21: CV based billing system

After the billing start,s the detected products are automatically added to the list. As we automatically refresh, the products will disappear from the list if we remove them from the checkout area. In order to achieve the batch billing system, we introduced the locking mechanism. When "Lock items" is pressed, it locks the current items to the top and highlights them in green. We can see in the Figure

above that the items highlighted in green are locked. When we finish billing, only the locked items in the list will be billed.

Apart from the locked items, we can see the same items in a non-highlighted state under it. The application is considering the item as a new entry; if we remove the item from view, it will automatically be removed. Furthermore, if we press stop billing in this state, it will only bill the locked items.

4.5.3.3 Receipt Generation

A receipt is generated after "Stop Billing" is pressed. All the locked items are taken from the list and presented in a POS-style receipt format in a .txt file. This is just for the representation of the receipt. In a practical application, we plan to convert the .txt format to POS commands in order to print the receipt via a POS thermal printer. The prices for the individual items are stored in a local CSV file. One can change the values from the file in order to update the prices.

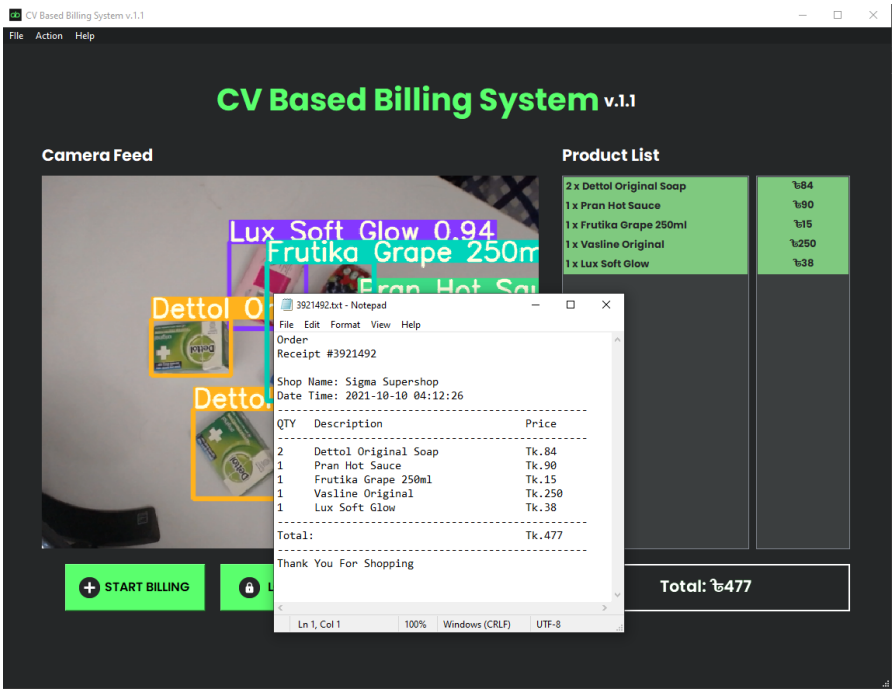


Figure 4.22: Generated Receipt

CHAPTER 5

RESULT AND ANALYSIS

In this chapter, we aim to discuss the results of our implementation and analyze its performance in various scenarios and test setups. We also discuss how we finalize the model to use as the backend of the billing system application.

5.1 Dataset Analysis

Let's analyze the bounding box data from the last version of the dataset. We use correlograms as a method of analysis. Seaborn pair plots [102] create correlograms by creating a histogram plot with each pair of parameters. The plots at the diagonal are simple distributions.

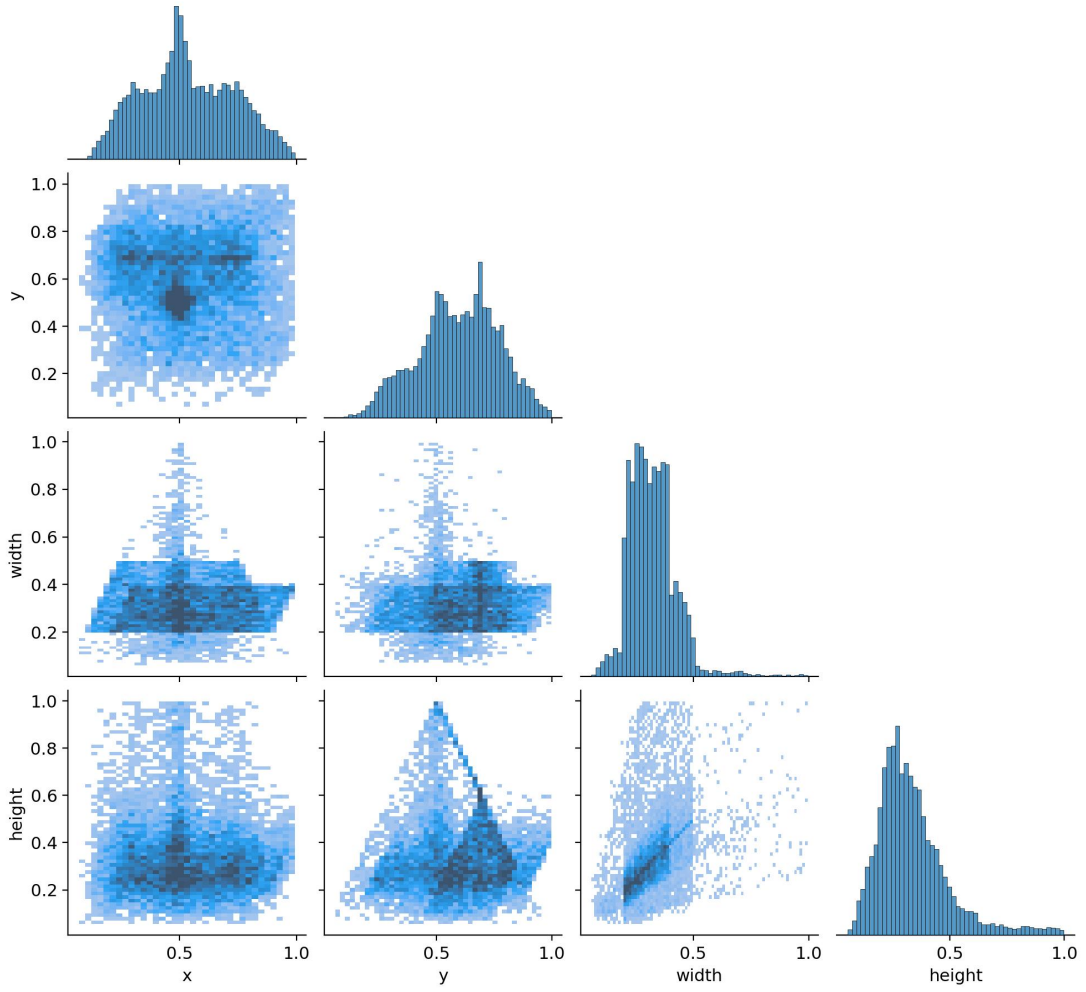


Figure 5.1: Correlogram of the dataset

These are pair plots of the bounding box parameters of each of the object instances. The only graphs of interest are xy and height-width graphs. It gives us an idea about the distribution of the objects throughout the images. From xy figure, we can infer that a lot of the images have x and y coordinates near the center of the image as there is a very dense spot in the middle of the plot. This is expected as there are a lot of daraz review images in our dataset which are just the picture of the single product. From the height-width plot, we can infer that there is a good amount of square bounding boxes in the images as there is a dense diagonal line that represents a linear increase in height with an increase in width.

5.2 Evaluation Metrics

We will discuss some popular metrics for object detection in this section. There are two parts of the object detection process, the first is object localization and the other is classification. One of the used evaluation metrics for localization is Intersection over Union (IoU)

5.2.1 Intersection Over Union (IoU)

IoU is the ratio of the area of intersection between the labeled bounding box (Bl) and the predicted bounding box (Bp) and the area of the union of them. A higher value of IoU represents a more correct localization of an object. For an application, usually, a threshold value is selected and if IOU is greater than that threshold it is considered to be correct [103].

Figure 5.2: Intersection over union

In the Figure5.2, the green bounding box is the ground truth or labeled bounding box and the yellow one is predicted by the model. The shaded area represents the intersection between them. IoU is defined by the ratio of the shaded area and the sum of the areas of the two bounding boxes.

5.2.2 Evaluation of Object Classification

After localization comes the classification of objects. Before we begin some common concepts should be discussed, True Positive: If the object has been correctly predicted in a scenario where it is present. True Negative: If the object has been correctly not found in a scenario where it is absent. False Positive: If the object is has been detected in a scenario where it is absent. False Negative: If the object has not been detected in a scenario where it is present.

Confusion Matrix:

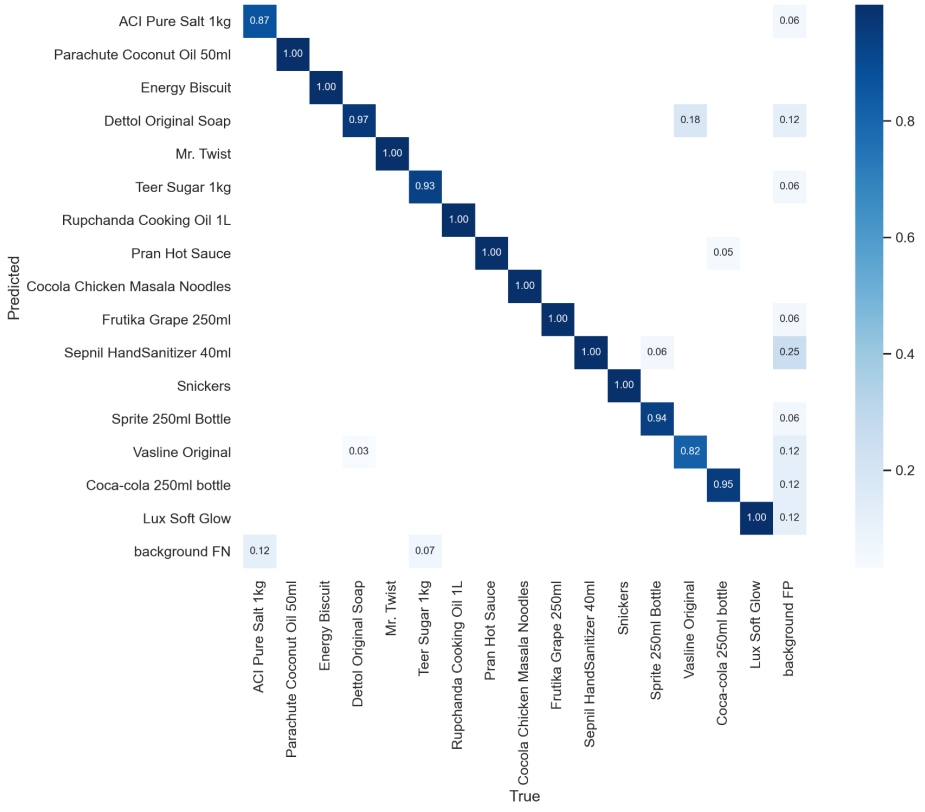


Figure 5.3: Confusion Matrix

A confusion matrix depicts the points where a model is confusing between classes. It is a nxn matrix with true and predicted results at perpendicular and lateral sides. The diagonal of the matrix represents the true positive predictions. Other points of the matrix show where the model has misclassified. For example in the figure above, 18% of Vaseline Orignal instances were confused with Dettol Soap. It is a very effective way of evaluating a model on a dataset [104].

Precision:

Precision is the measurement of how many positive predictions are actually correct. It focuses on the correctness of positive detection which can be a good metric for cases where a false positive can be more troublesome than a false negative.

$$Precision = \frac{TP}{FP + TP}$$

Recall: Recall is the measurement of how many positive predictions were done in scenarios where the object was actually present. It is a good metric where false negatives are more troublesome.

$$Recall = \frac{TP}{FP + FN}$$

F1 Score: It is the combination of Precision and Recall scores. It is a good choice for tasks that are sensitive to both false positives and false negatives.

$$F1Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

P-R Curve & Average Precision(AP): Precision-Recall Curve can be seen as a balance between the two metrics. For a good model, the precision should increase with the increase in recall. The area under the curve is a good metric for evaluation. One of the techniques for calculating the area under the curve is done by 11 point interpolation method. The precision value for 11 uniformly distant recall levels are averaged to get the value for AP [105].

$$\mathrm{AP}_{11} = \frac{1}{11} \sum_{R \in \{0, 0.1, \ldots, 0.9, 1\}} P_{\mathrm{interp}}(R) \tag{5.1}$$

where,

$$P_{\mathrm{interp}}(R) = \max_{\tilde{R}:\tilde{R} \geq R} P(\tilde{R}) \tag{5.2}$$

P(R) where the value of recall is greater than R is used for AP calculation instead of precision at R value.

Mean Average Precision(mAP): mAP is the mean of the AP for all classes in a dataset, it is one of the most used metrics for object detection evaluation.

$$\mathrm{mAP} = \frac{1}{N} \sum_{i=1}^{N} \mathrm{AP}_i \tag{5.3}$$

Most of the renowned competitions use this metric as their main metric such as the PASCAL VOC Challenge, COCO Object detection challenge [106] and the Open Images challenge [107].

Confidence Score: It is the probability of the predicted object being in the bounding box.

Accuracy: It is defined by the total correct predictions by total predictions. It is not an ideal metric for object detection as the dataset may not be balanced

5.3 Test setups

We have created a total of 3 test setups each focusing on different aspects of the object detection problem.

Test Set 1: This test set focuses on the highly dense placement of the products and how our models perform on them. We are using 30 images containing a total of 351 object instances. This set of images are arranged in two parts, front sided images and back-sided images. This is used for evaluation under 2 different lighting conditions. One where it is well lit and another where there is not enough light.

The dark setting was created artificially by lowering the brightness of the images in order to keep other factors unchanged.

The distribution of front-facing products in the set are shown in the pie chart below:

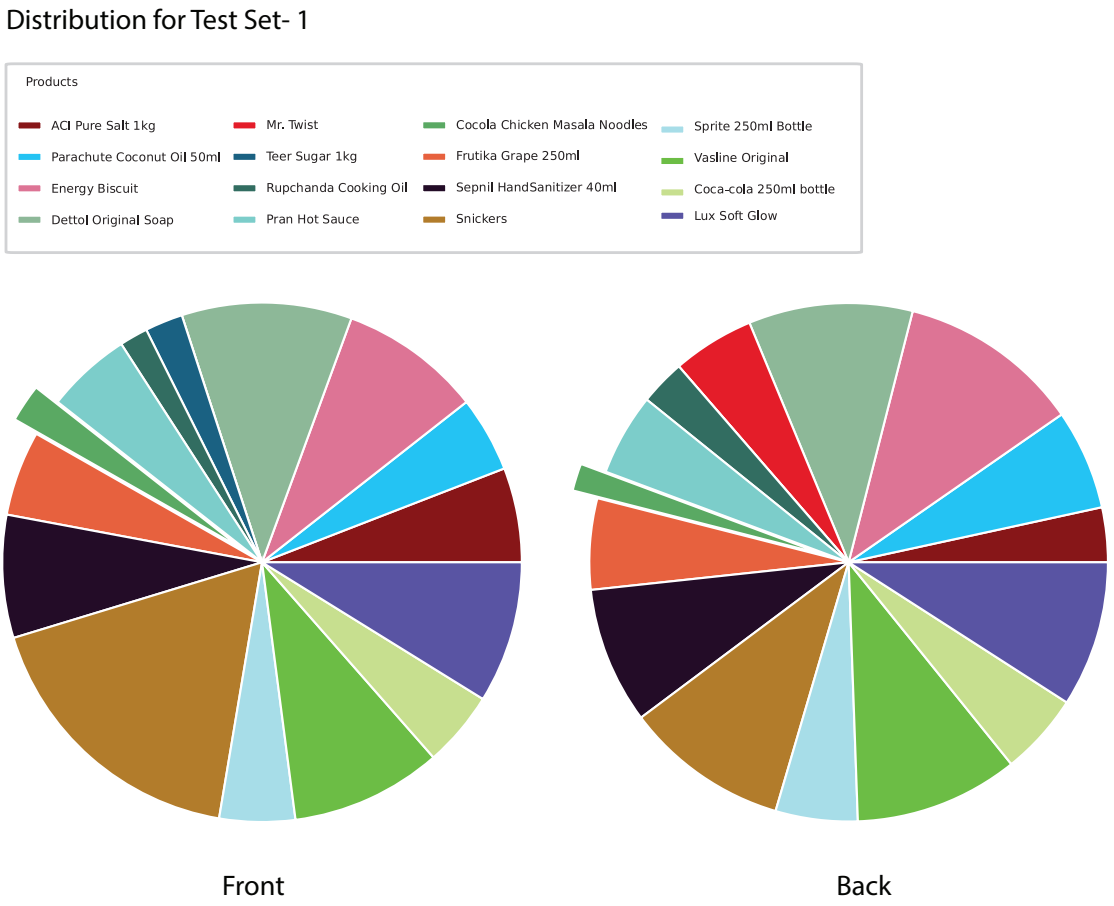


Figure 5.4: Distribution of Test Set -1

Test Set 2: This test set focuses mainly on real-time detection. This is a 1:30 min long clip of a person using the system to bill products. We take several frames from different locations of the clip to evaluate our models.

Test Set 3: This test set focuses mainly on the detection of products on different backgrounds; it contains 818 object instances of 26 classes spanning over 216 images which is a 10% split of the original dataset. The products on this set are on various backgrounds allowing for a robust background-independent test.

The distribution of products in the set are shown in the pie chart below:

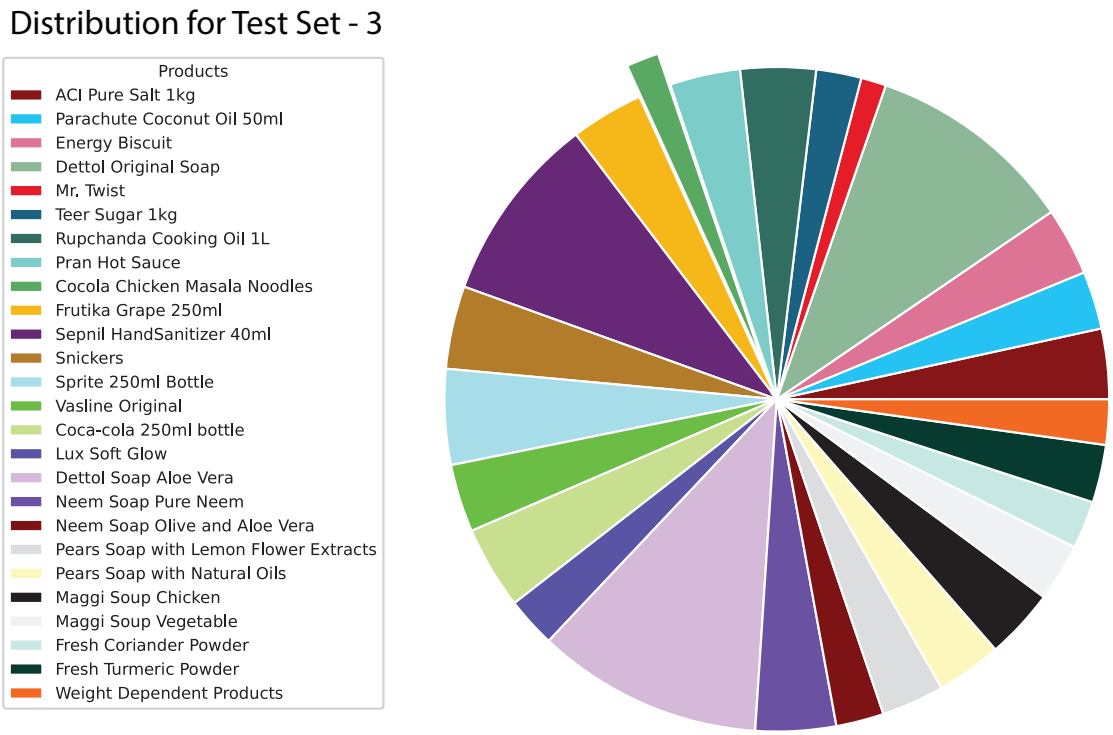


Figure 5.5: Distribution of Test Set -3

Test Set 4: This set focuses mainly on very similar looking products. Mostly variations of the same product which are difficult for the models to detect. The set consists of very densly packed 507 object instances of 10 classes spanning over 30 images.

The distribution of products in the set are shown in the pie chart below:

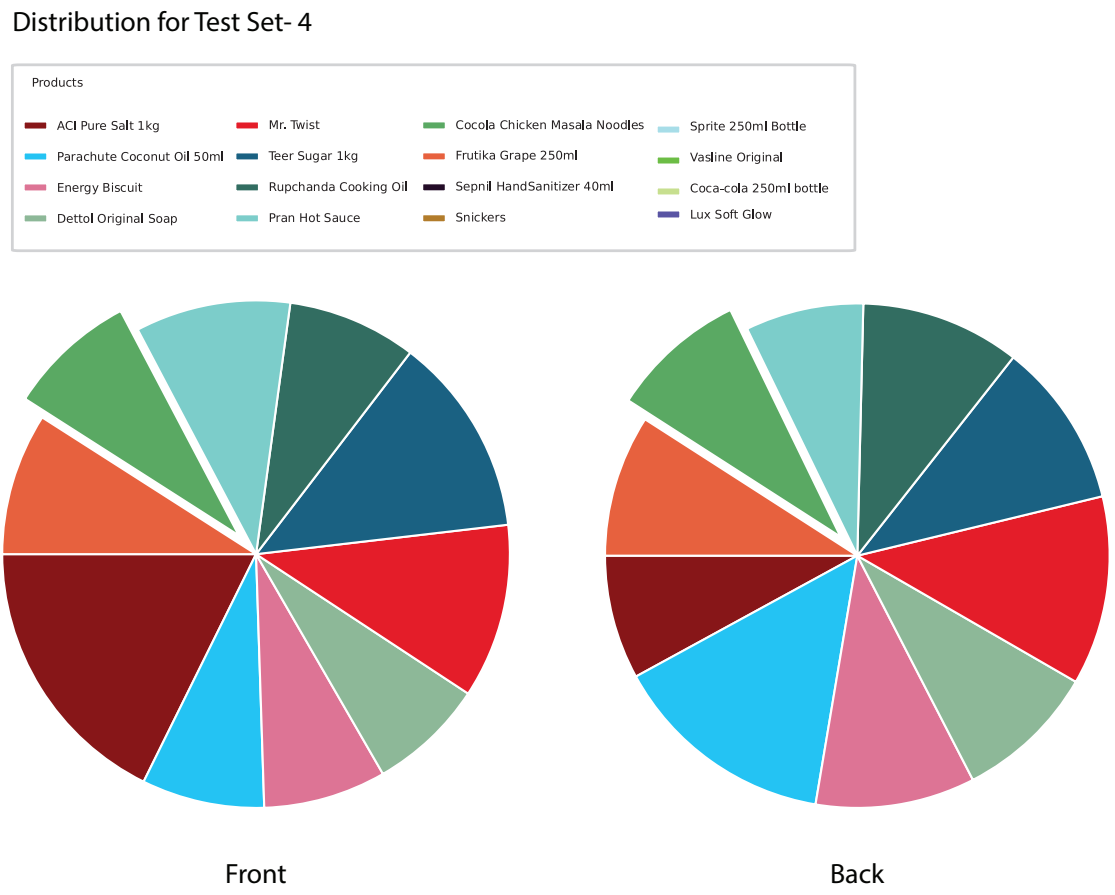


Figure 5.6: Distribution of Test Set -4

5.4 Models for Evaluation

In different stages of development, we have trained different models either based on different model architectures or different states of the dataset. So far, we have only talked about our final result. In this section, we shall specify all of the models we used for testing:

1. Synthval_Small: Yolov5s model. Our validation set contained synthetic images during this phase, and most of the dataset was full of individual product images.

2. NoStack_Small: Yolov5s model. We removed synthetic images from our validation set.

3. Stacked_Small: Yolov5s model. We added more densely packed multi-product images to our dataset

4. Stacked_Medium: Yolov5m model. We added more densely packed multi-product images to our dataset

5. Aruco_Medium_300: Yolov5m model trained upto 300 epochs on Version-5 of the dataset with 3 anchors per output layer.

6. Aruco_Large: Yolov5l model trained upto 100 epochs on Version-5 of the dataset using slightly more augmentation and no anchors per output layer.

7. Ensemble_Aruco_Large_Medium: Ensemble made with Aruco_Medium_300 and Aruco_Large models.

5.5 Training Results

Here we showcase the training statistics of the stacked medium model. There are a couple of parameters here.

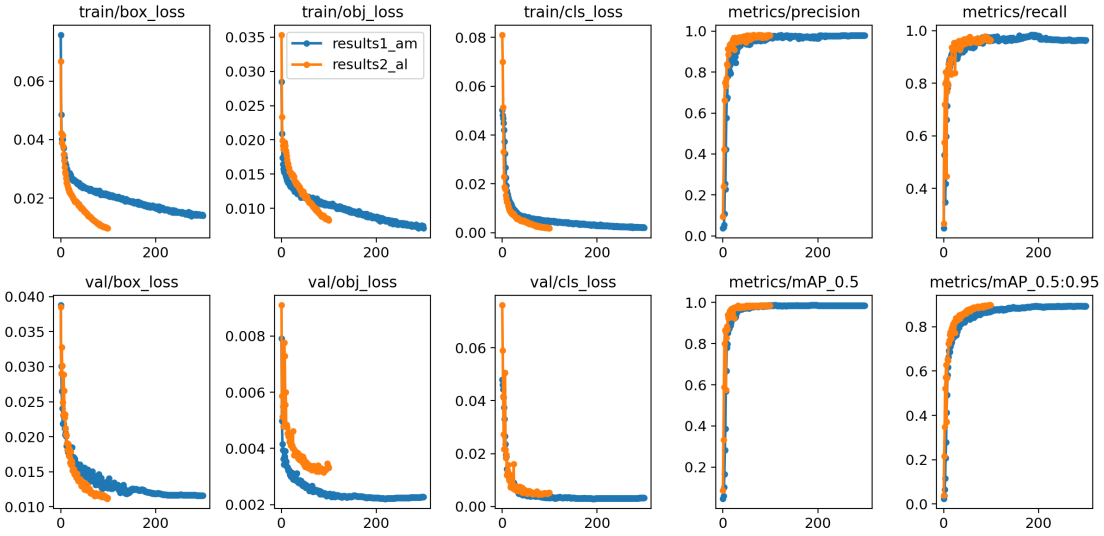


Figure 5.7: Training Statistics for Aruco_Medium_300 (Blue) and Aruco_Large (Orange) models

The Box metric is actually the coordinate loss; it helps the model predict better boxes. The objectness actually represents the objectness loss, minimizing it helps the model improve IoU. The Classification loss does a similar thing in the case of improving classification. These curves are plotted for both the training set and the validation set. We can notice that the losses are slowly decreasing as epochs increase. Precision and Recall curves are also given. We can notice that both of them increase with epoch numbers and reach a saturation level near the end. The mAP curve also follows a similar pattern, it starts off a bit zig-zag, but it eventually reaches a high value and saturates. The training statistics for most of the models look similar; that is why we are only showcasing our overall best models.

We noticed that the large model tends to overfit faster. As we can see near the end of the 100 epoch run the validation objectness loss started to rise again while the training loss was low, which is an indication of overfitting. We took the weights of the best epoch in order to mitigate this. This is the reason why we trained the large model for only 100 epochs while we trained the medium model for 300 epochs.

5.6 Results on Densely Arranged Objects

The models mentioned above are all evaluated on Test set - 1. We can illustrate the results for each model using confusion matrix below:

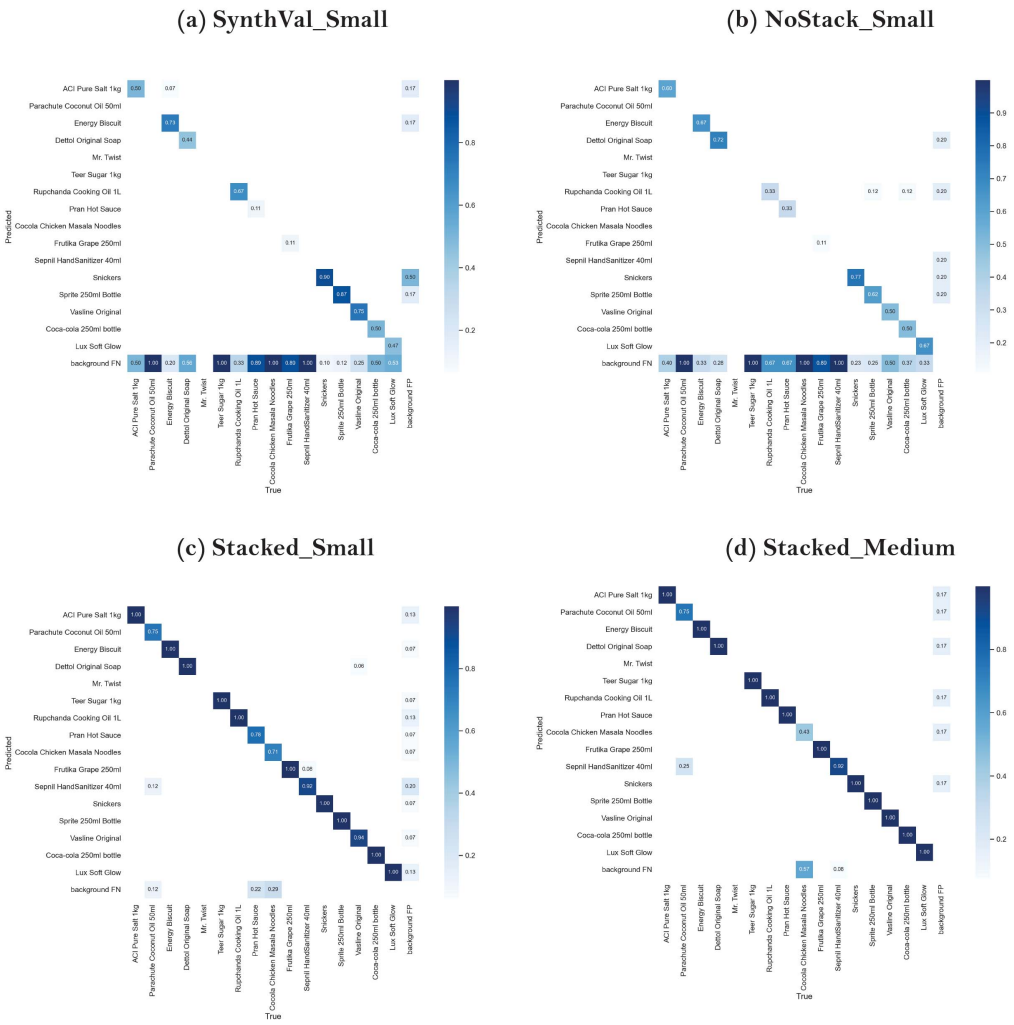


Figure 5.8: Confusion Matrices for Front facing products in Test set-1

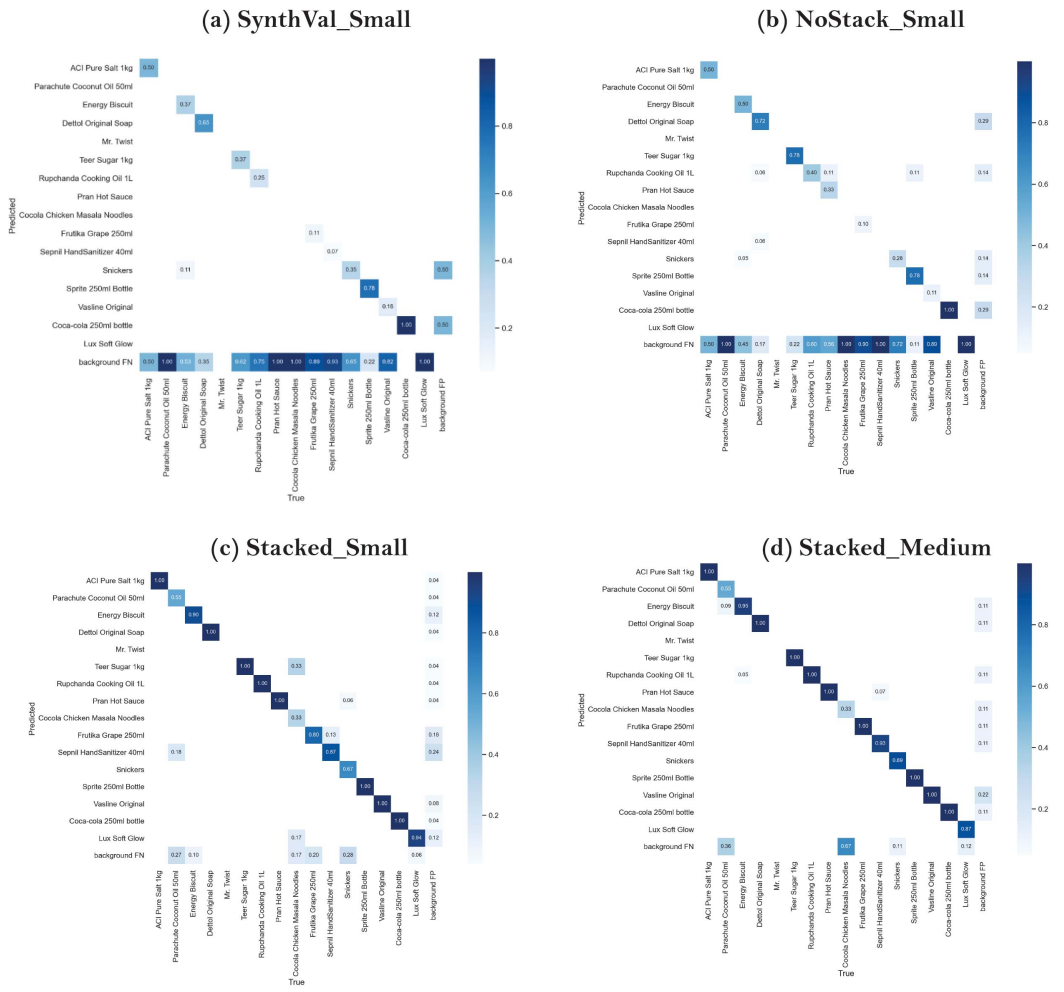


Figure 5.9: Confusion Matrices for Back facing products in Test set-1

If we notice the matrices (a) and (b) for both figures (front and back) we see a noticeable improvement. This is the result of removing all the synthetic data from our validation set. The first model was fitting more towards synthetic images as they were in the validation set. Thus reducing its accuracy on real images that are often not as perfect in terms of background separation as synthetic ones.

More images of the densely packed product were added between (b) and (c). Here we can see substantial improvement in detection It has gotten very good at

correctly classifying the products at this point.. Also between (c) and (d) can observe more improvement as we move to a larger model which is Yolov5.

There is one important thing to mention here is that from the Pie charts of the test set 1 we can see that there is no representation of Mr.Twist in this set, hence the discontinuity in the diagonal of the matrix. Also, the reason for Cocola Chicken Noodles to have the lowest score across the models is that the item has a very low presence in this set. We can also notice that the performance of the models is better on the front labels than on the back. The prime reason for this is the fact that data of front and back images of each product are not distributed uniformly in the dataset. The front-facing images are most likely to be greater.

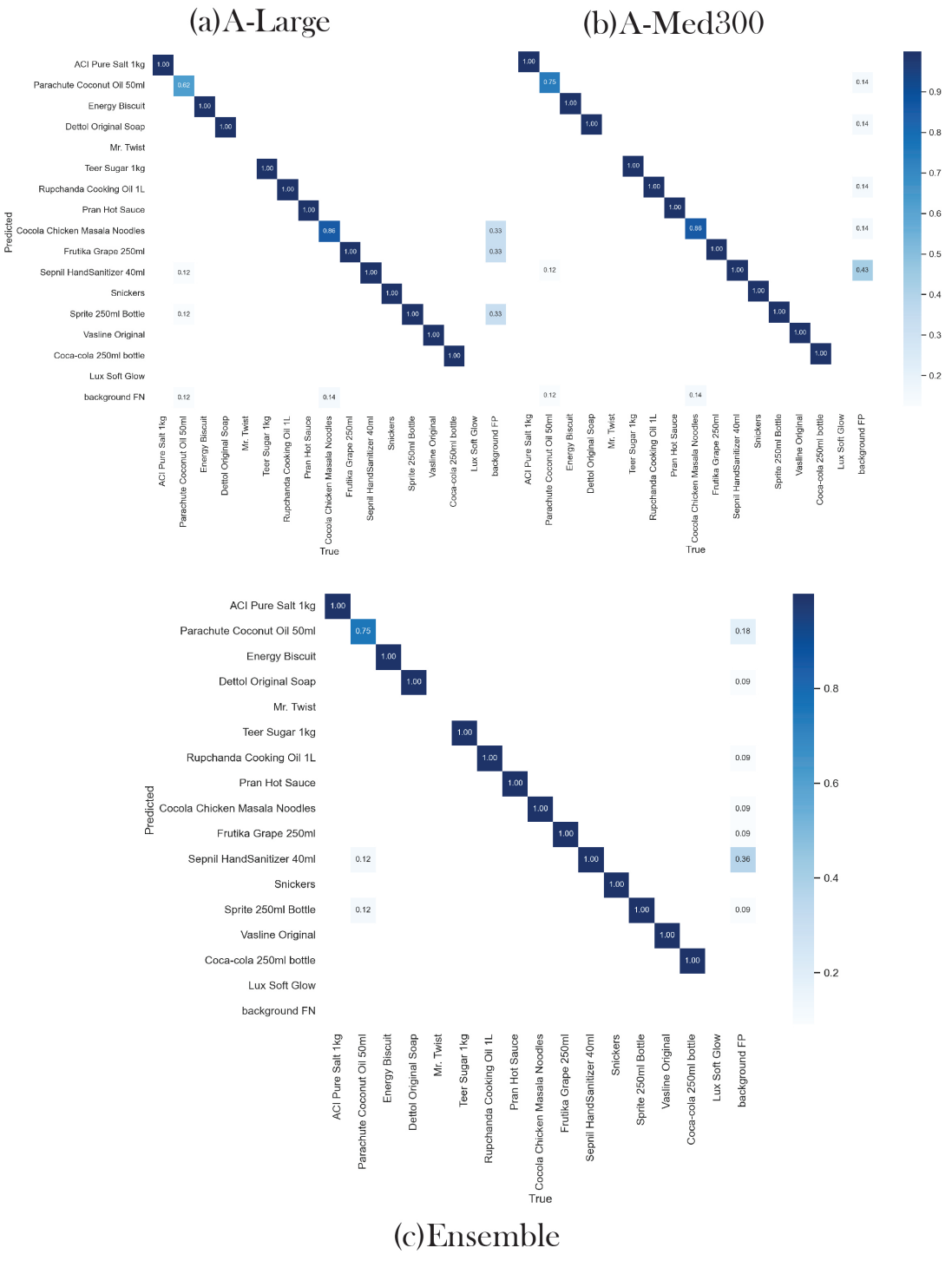


Figure 5.10: Confusion Matrices for Front labels of Test Set - 1 (a) Aruco_Large (b) Aruco_Medium_300 (c)Ensemble_Aruco

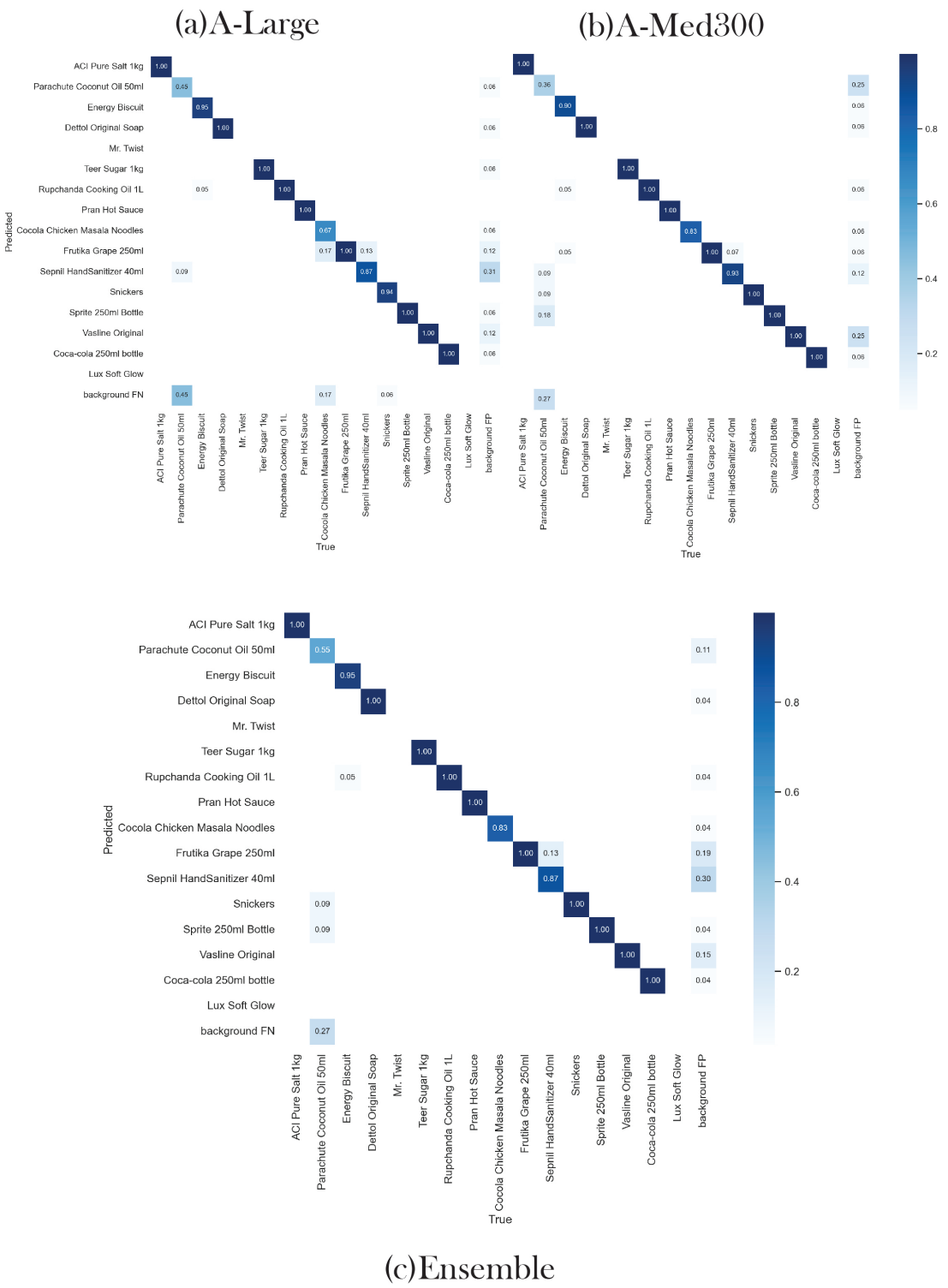


Figure 5.11: Confusion Matrices for Back labels of Test Set - 1 (a) Aruco_Large (b) Aruco_Medium_300 (c)Ensemble_Aruco

In Figure 5.10 we notice that the new models have performed much better than the old ones. The Medium_300 model shows false positive results over a larger class variation but the Large model shows false positives over a smaller range of classes but the number of false positves are huge. But the Ensemble model was able to show the most overall optimal performance esp. minimizing confusion within classes.

In Figure 5.11 we notice similar outcomes. Here the ensemble model was also able to lower the number of false postives. But we notice a rise in confusion for Parachute Oil as the larger models are tending to overfit on this particular class due to the bottle images from daraz not being uniform across this class.

5.7 Results in Different Lighting Conditions

mAP scores for the 4 models are shown in the chart below for both front and back labels, each of which is in bright and dark settings separately.

In the Figure 5.12 and 5.13, each group of 8 bars represent mAP scores for each model. The first subgroup group of 4 bars represent performance on front labels and the other 4 on back labels. First two bars for each subgroup represents mAP@0.5 for bright and dark lighting conditions, the other two represents mAP@0.5:0.95 scores for bright and dark lighting conditions within the given test set slice.

If we analyze the information in the chart, 3 points of improvement can be noticed. Firstly, we can see that the overall mAP scores improved a bit after the removal of synthetic data from the validation set. Secondly, the scores significantly improved after the inclusion of the 55 highly dense images of products in the dataset. Thirdly, we can notice significant of improvement after changing to a larger model and training for more epochs. The Ensemble shows overall best performance however the change is quite insignificant and marginal compared to the Aruco_Medium_300 model.

The scores for the back labels are somewhat lower than the front labels scores. There are two reasons that might have caused this. The back labels of most

products don't have enough recognizable features, also the number of backside images are low in the dataset as most authentic images come from Daraz reviews.

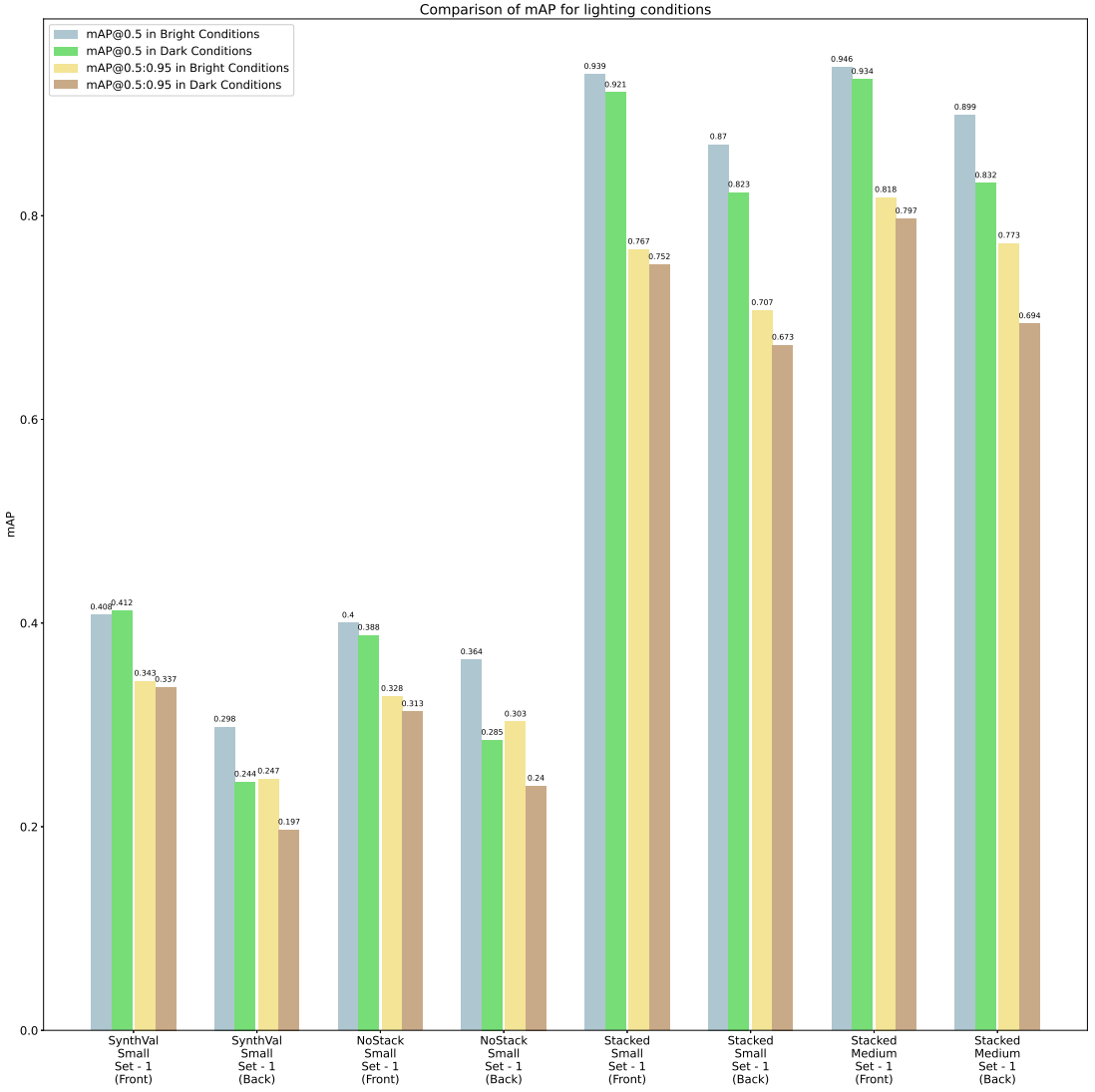


Figure 5.12: Comparison of mAP in different lighting conditions

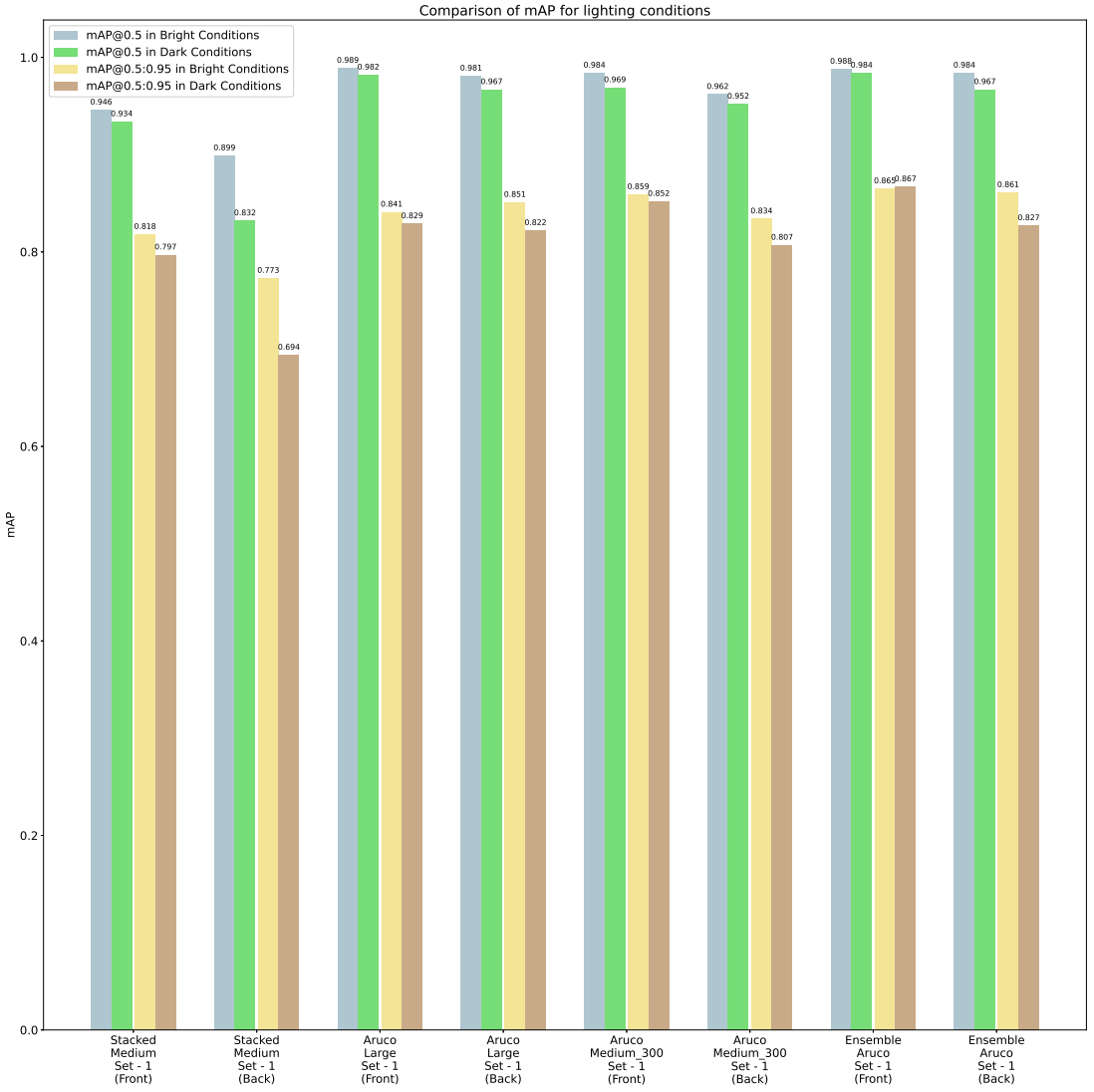


Figure 5.13: Comparison of mAP in different lighting conditions (New Models)

5.8 Analysis of Real-time detection

Even though it's difficult to put the results of real-time detection into a representable form on paper. We make an attempt to evaluate performance based on our Test Set - 2. We pick several frames from the video and evaluate performance based on confidence scores and the number of false positives and false negatives. This depicts how confidently the model is detecting the products in each scene. The scense are shown below

Figure 5.14: Selected Frames for the Test-2

The confidence scores and number of errors for each of these scenes are plotted in bar charts below. The evaluation of performance should depend on both of the parameters together.

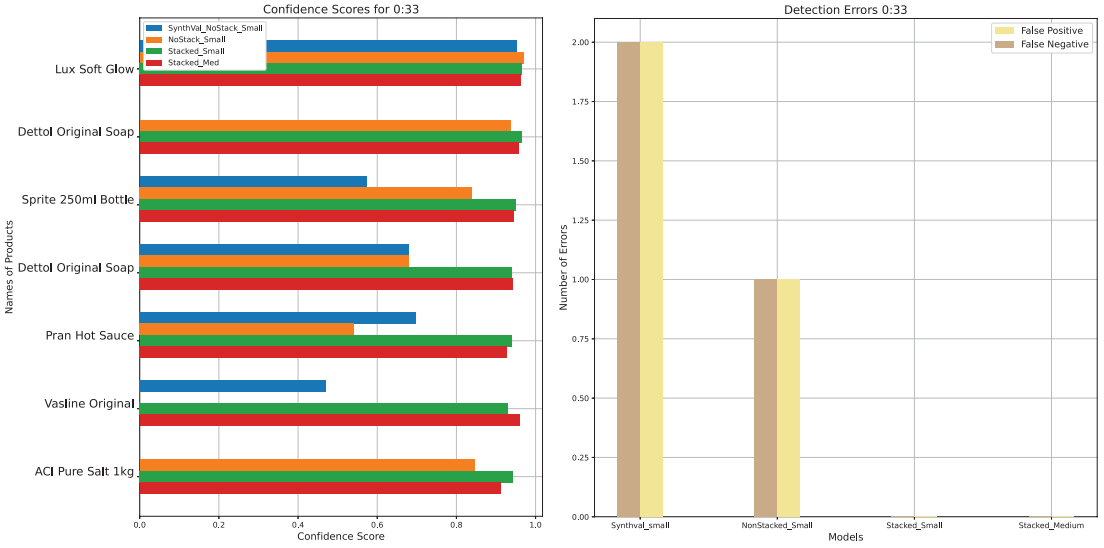


Figure 5.15: Analysis for time frame 0:33

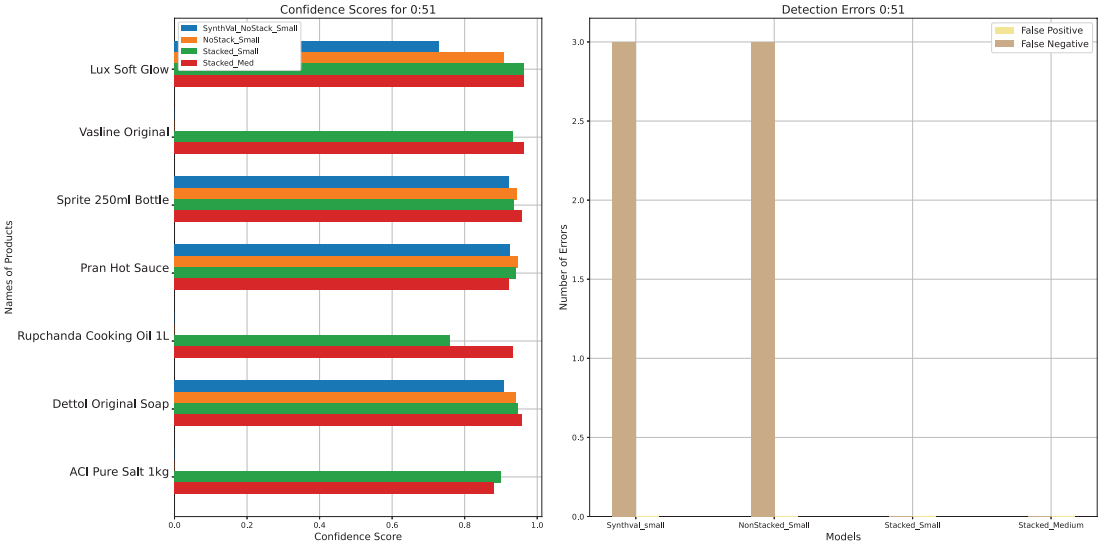


Figure 5.16: Analysis for time frame 0:51

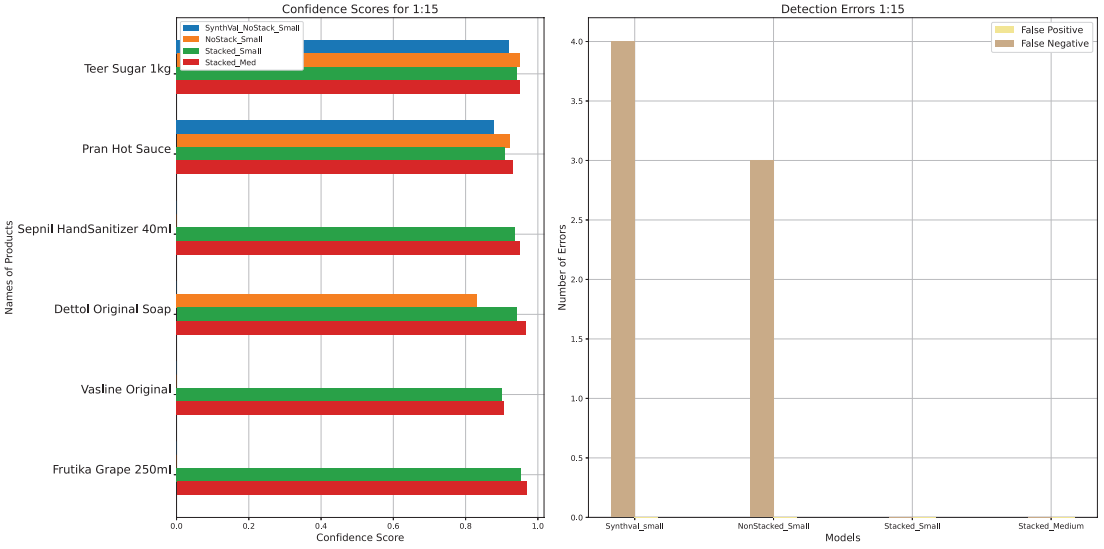


Figure 5.17: Analysis for time frame 1:15

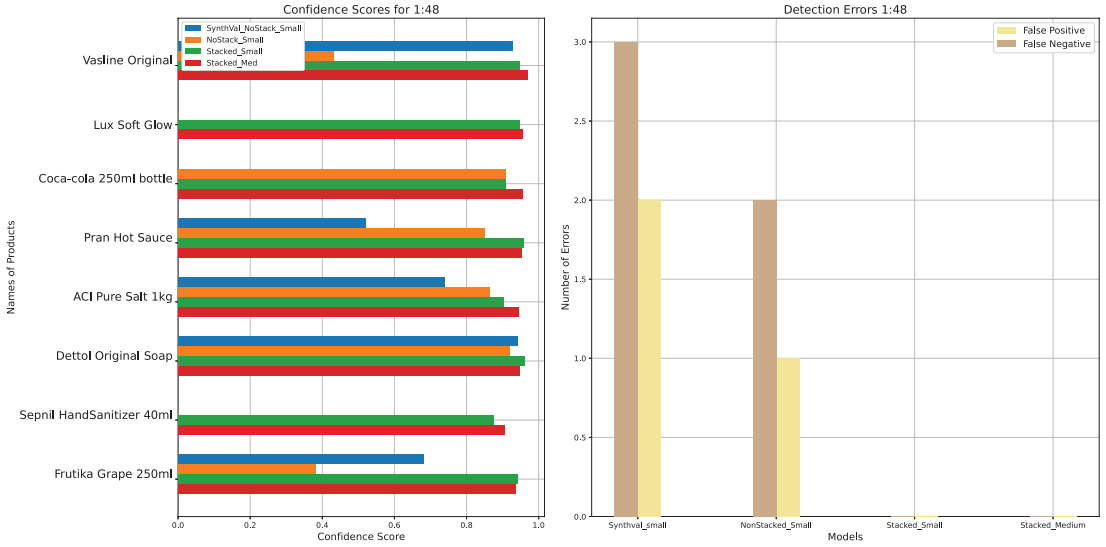


Figure 5.18: Analysis for time frame 1:48

From all the figures above, it is significantly noticeable that after adding the staked images to the dataset, real-time detection has improved. The confidence scores are consistent and there are zero detection errors in the selected time frames. The performance for both small and medium models are very similar. In a later section, real-time performance change due to hyperparameter evolution has been discussed. Note that we haven't shown results for our newer models and ensemble because they will perform very similarly to the Stacked_Medium model.

5.9 Analysis on Different Backgrounds

The confusion matrices for the model's performances on Test Set - 3 are shown in this section.

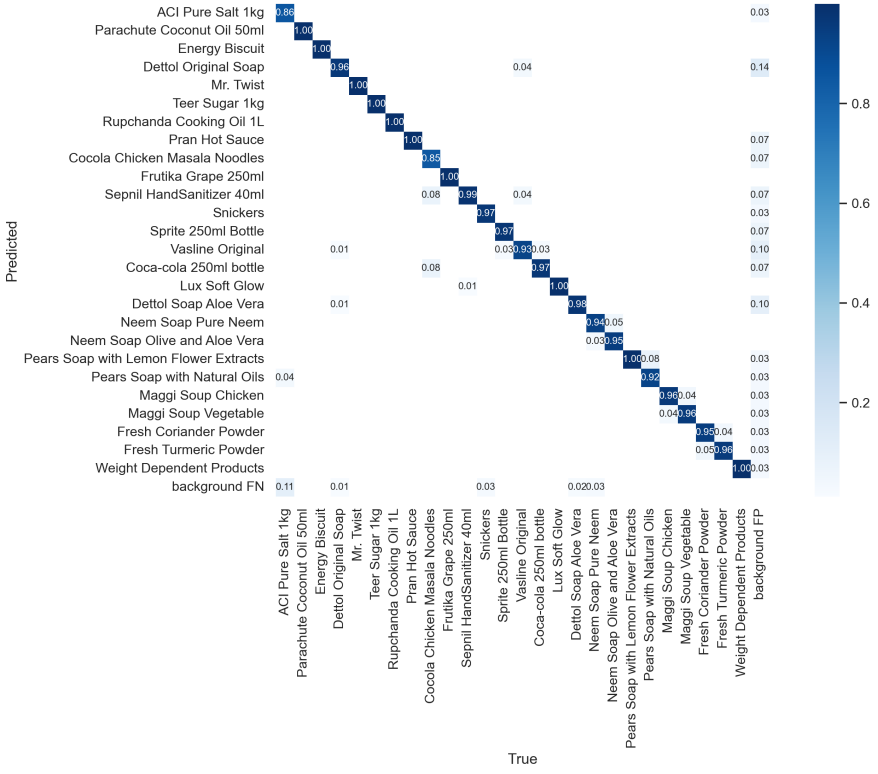


Figure 5.19: Confusion Matrices of Aruco_Medium_300 model on Test Set - 3

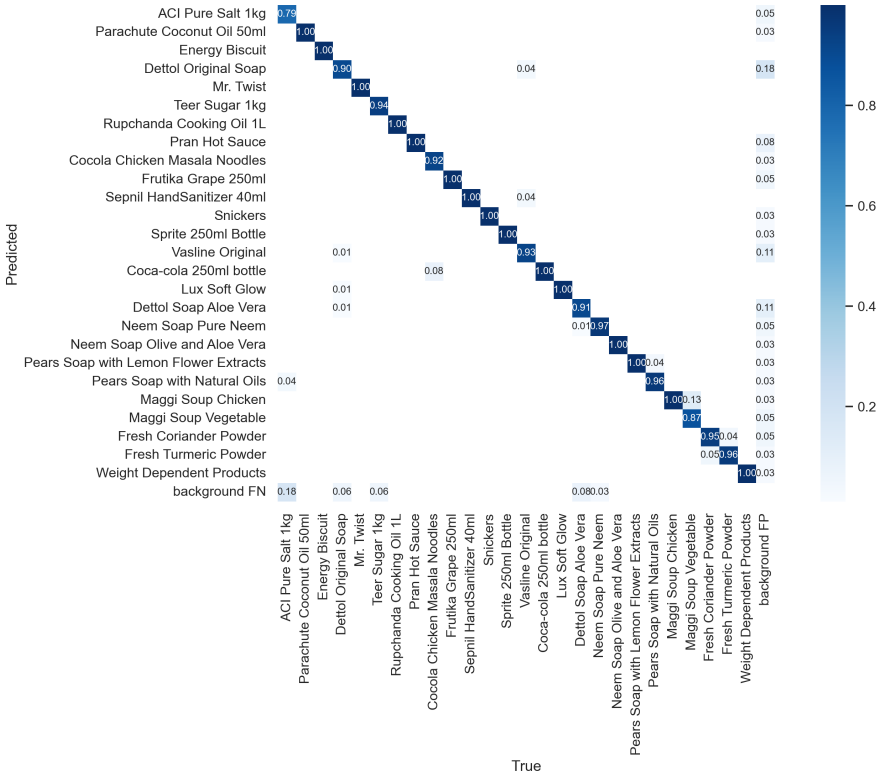


Figure 5.20: Confusion Matrices of Aruco_Large model on Test Set - 3

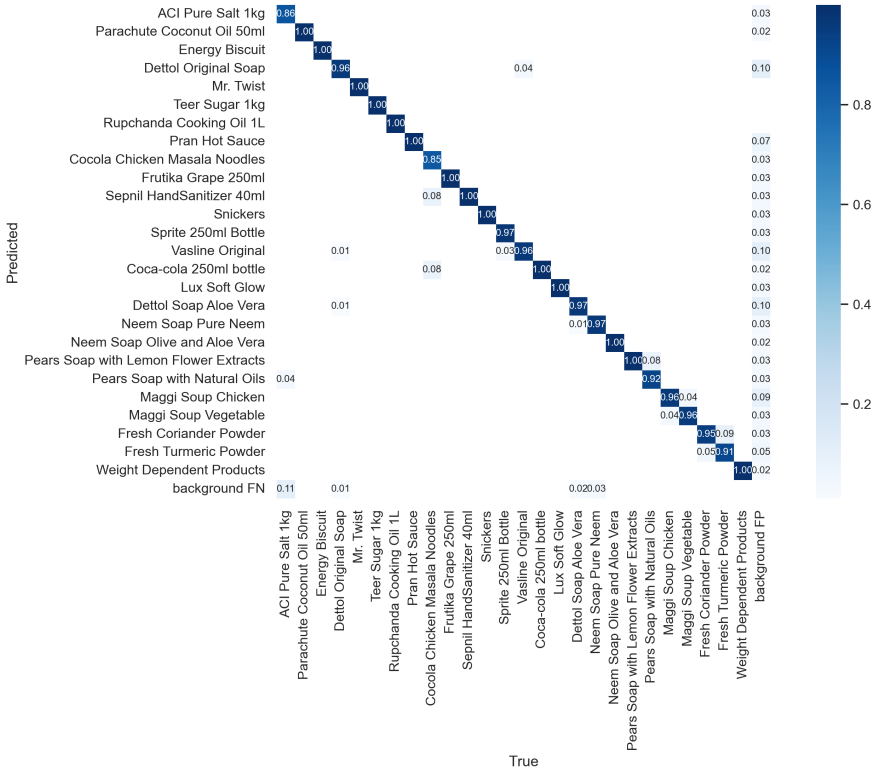


Figure 5.21: Confusion Matrices of Ensemble model on Test Set - 3

From the matrices above we notice slight improvement in performance with the ensemble model. It combined the good aspects of the other too model creating a small but noticeable improvement esp. in reducing confusion between products of the dataset.

5.10 Results after Data Augmentation

In Chapter 4, we have discussed the inclusion of synthetic data in our dataset. Here we compare the performance changes of the models based on mAP scores due to the addition of synthetic data. For simplicity, we have only compared the results of our Stacked_Medium model.

In the Figure 5.22, each group of 4 bars represent a test set. The first subgroup of 2 bars represent mAP@0.5 and the other subgroup represents mAP@0.5:0.95. The first bar and the second bar in each subgroup show performance without and with

data augmentation respectively. It can be noticed that mAP scores have improved almost 10% in all cases due to augmentation via synthetic data.

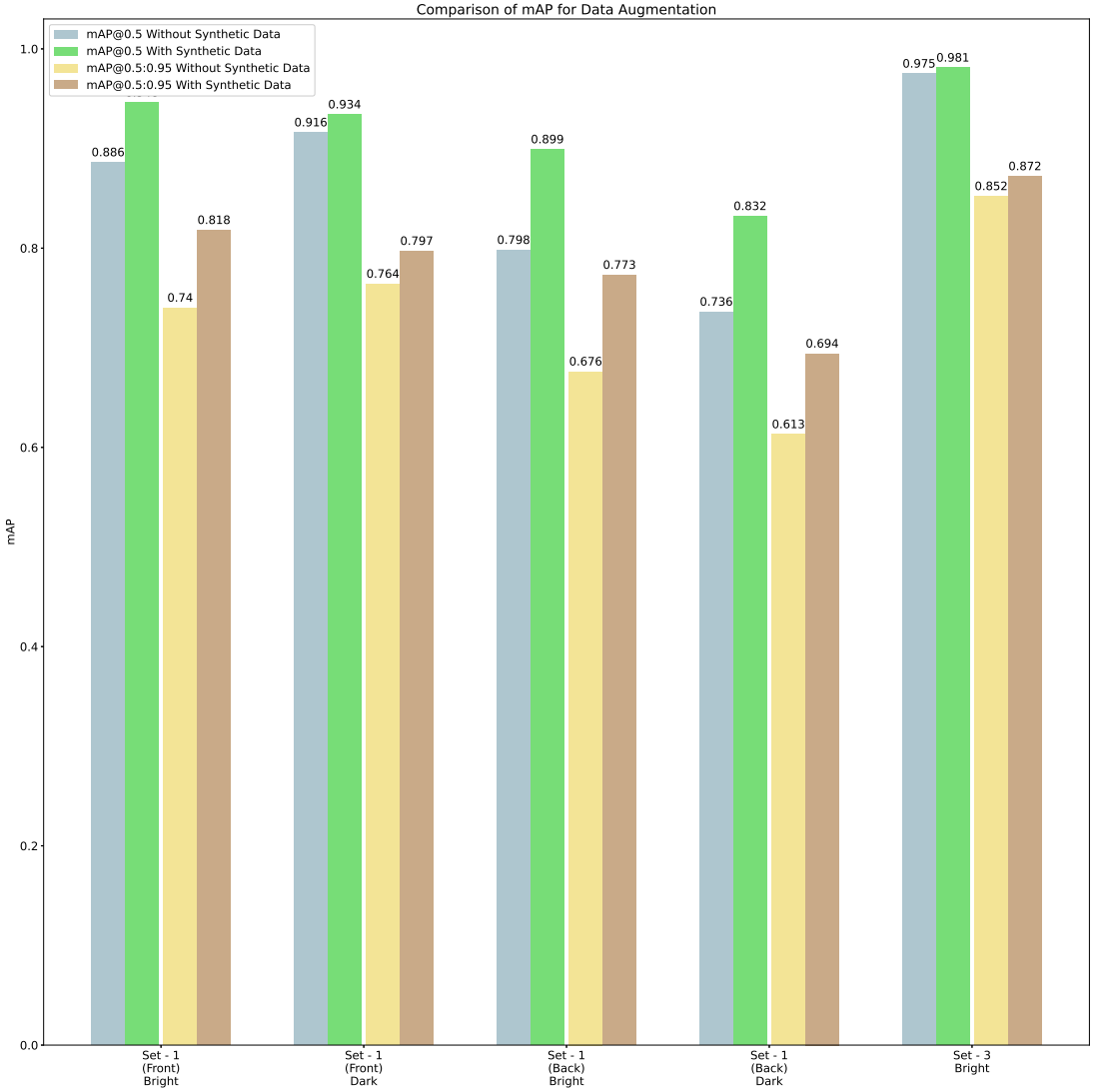


Figure 5.22: Comparison of mAP after Data Augmentation

5.11 Effects of Hyperparameter Evolution

In Chapter 4, we have also discussed hyperparameter evolution, here we discuss the results of the evolution and how it fairs against the default values we used before.

5.11.1 Values Obtained through evolution

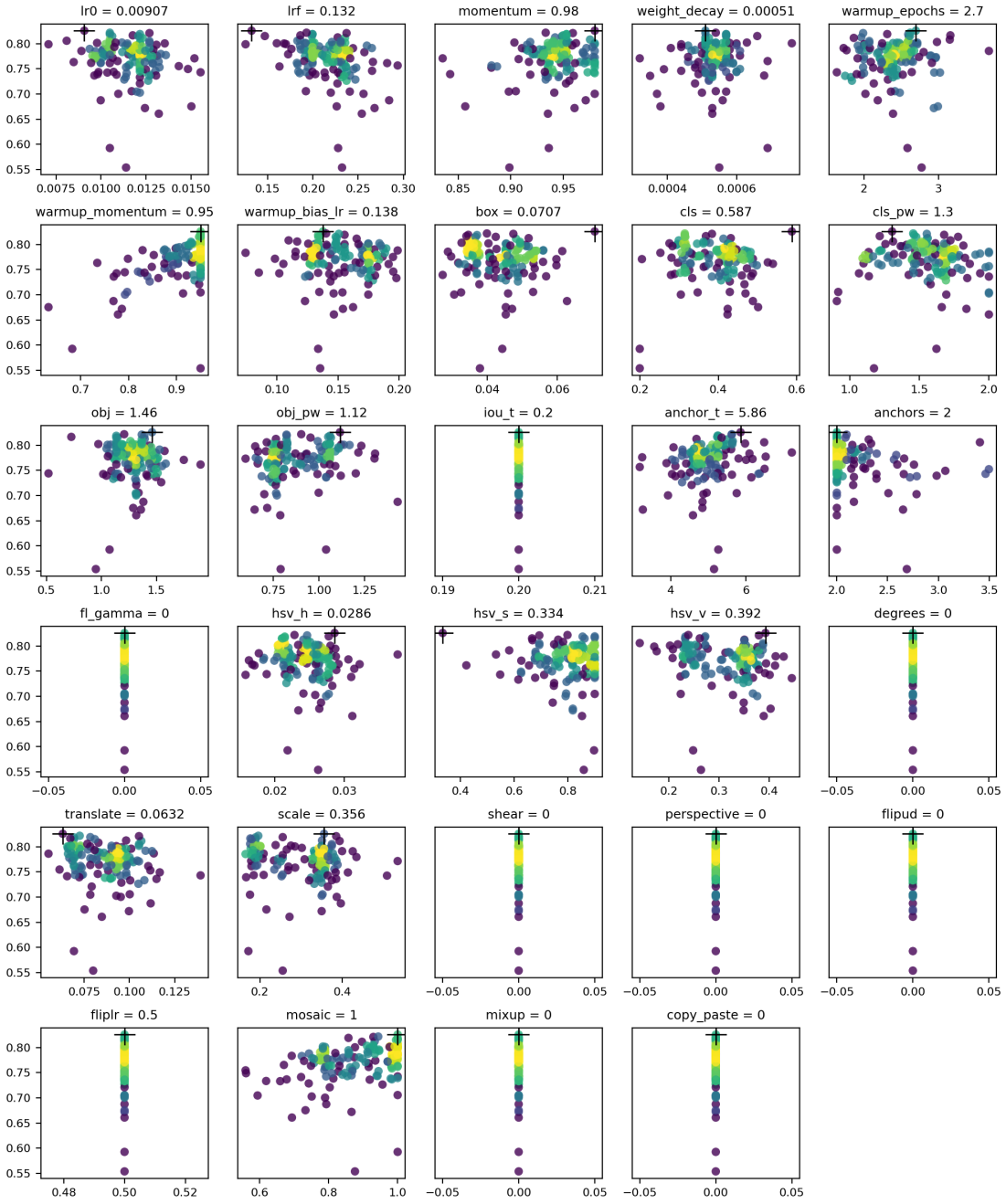


Figure 5.23: Distribution of Hyperparameter values for 150 generation of evolution

Based on the evolution, we obtained the best results on generation 11. Here are the hyperparameter values found in that generations:

lr0:	0.01269
lrf:	0.23471
momentum:	0.96013
weight_decay:	0.00035
warmup_epochs:	1.7472
warmup_momentum:	0.76925
warmup_bias_lr:	0.12307
box:	0.04871
cls:	0.38396
cls_pw:	2.0
obj:	1.17779
obj_pw:	1.03978
iou_t:	0.2
anchor_t:	6.14466
anchors:	2.72045
fl_gamma:	0.0
hsv_h:	0.02546
hsv_s:	0.83462
hsv_v:	0.44497
degrees:	0.0
translate:	0.11493
scale:	0.38548
shear:	0.0
perspective:	0.0
flipud:	0.0
fliplr:	0.5
mosaic:	0.91917
mixup:	0.0
copy_paste:	0.0

5.11.2 Performance comparison on test sets

We have trained the YOLOv5m model with both default and the generation 11 hyperparameters and tested them on our 3 test sets. The chart on Figure 5.24 is structured in the same ways as the previous one. We can observe that after hyperparameter evolution the performance has noticeably dropped on this test set. Each generation had been trained for 10 epochs, which may be the cause for these unsatisfactory results. 10 epochs of runtime cannot account for the scenario of 100 epochs, so the performance didn't scale proportionally. However, we do notice some improvements in real-time detection. On our Test Set -2, the model performs similarly well comparing to our model with default hyperparameters and improves on the consistency of detection. The detections for the evolved hyperparameters are much crisper.

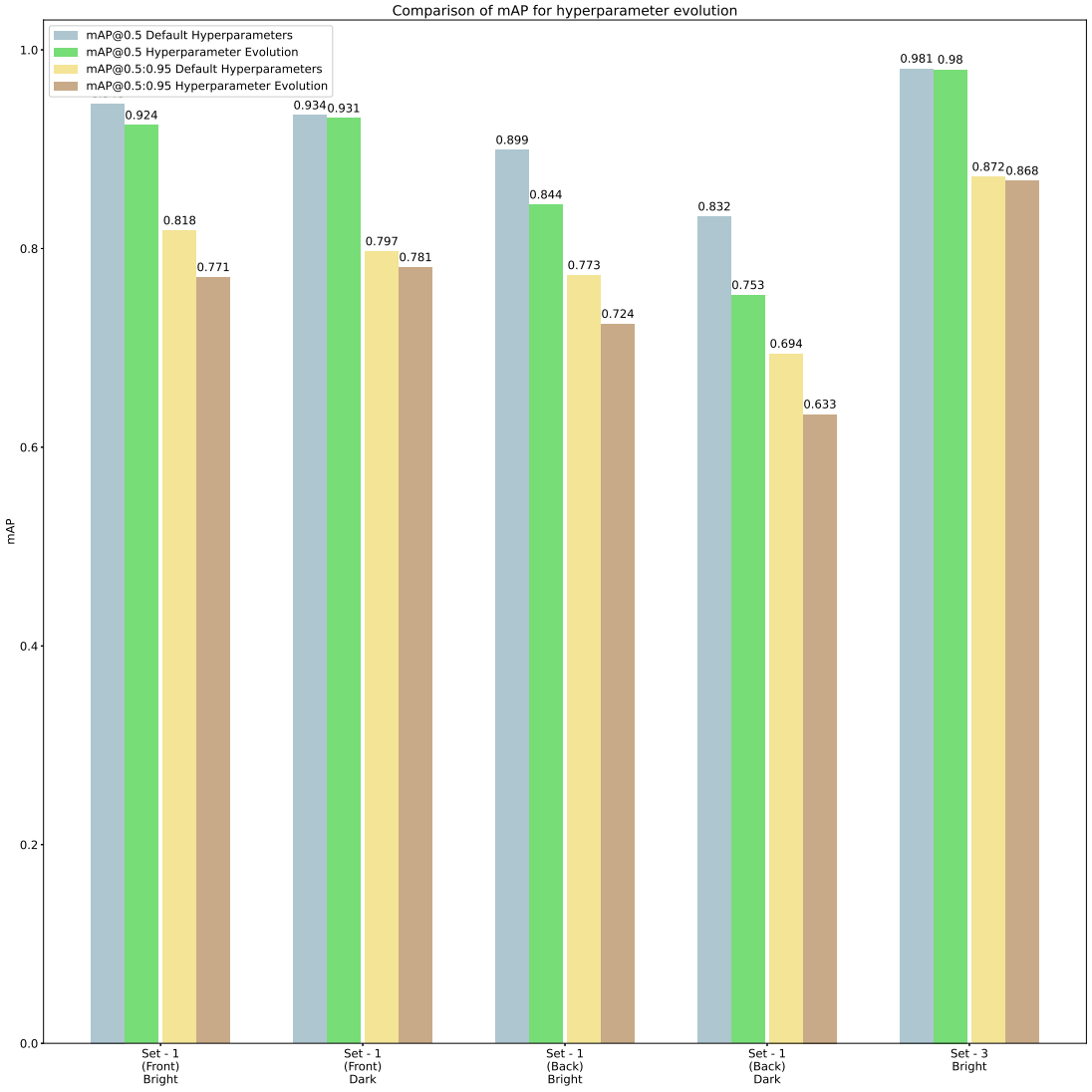


Figure 5.24: Comparison of mAP after hyperparameter evolution

In the Figure 5.25, we compare some frames of Test Set - 2. Frames 109-1019 are shown in one frame intervals. When the “Pran Hot Sauce” is placed into the scene the default hyperparameter model detects an additional “Rupchanda Cooking Oil” (lime green label) on frames 1013-1017 (marked with red circles). This is most likely due to the hand of the person and the moving bottle. But we notice no such glitches in the detection of the model after hyperparameter evolution.

Frame Anaylsis for Test Set 2 after Hyperparameter Evolution

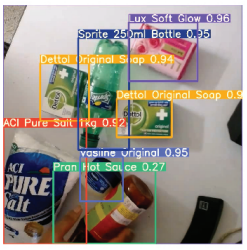
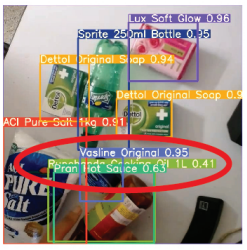
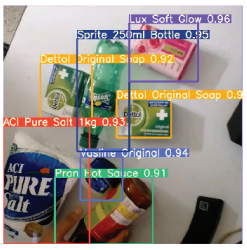
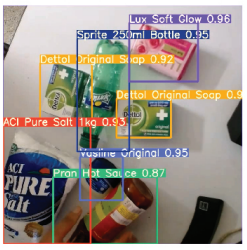
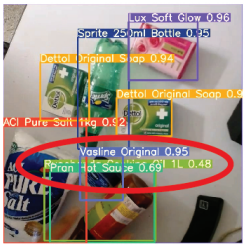
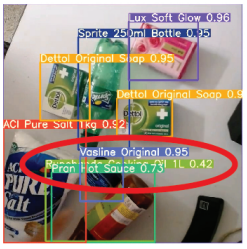
Frame	1009	1011	1013
Default Hyperparameter			
After Hyperparameter Evolution			
Frame	**1015**	**1017**	**1019**
Default Hyperparameter			
After Hyperparameter Evolution			

Figure 5.25: Frame Analysis for Test Set 2 after Hyperparameter Evolution

5.12 Analysis on Similar Looking Products

The confusion matrices for models Aruco_Medium_300, Aruco_Large, and Ensemble_Aruco on Test Set - 4 are shown below:

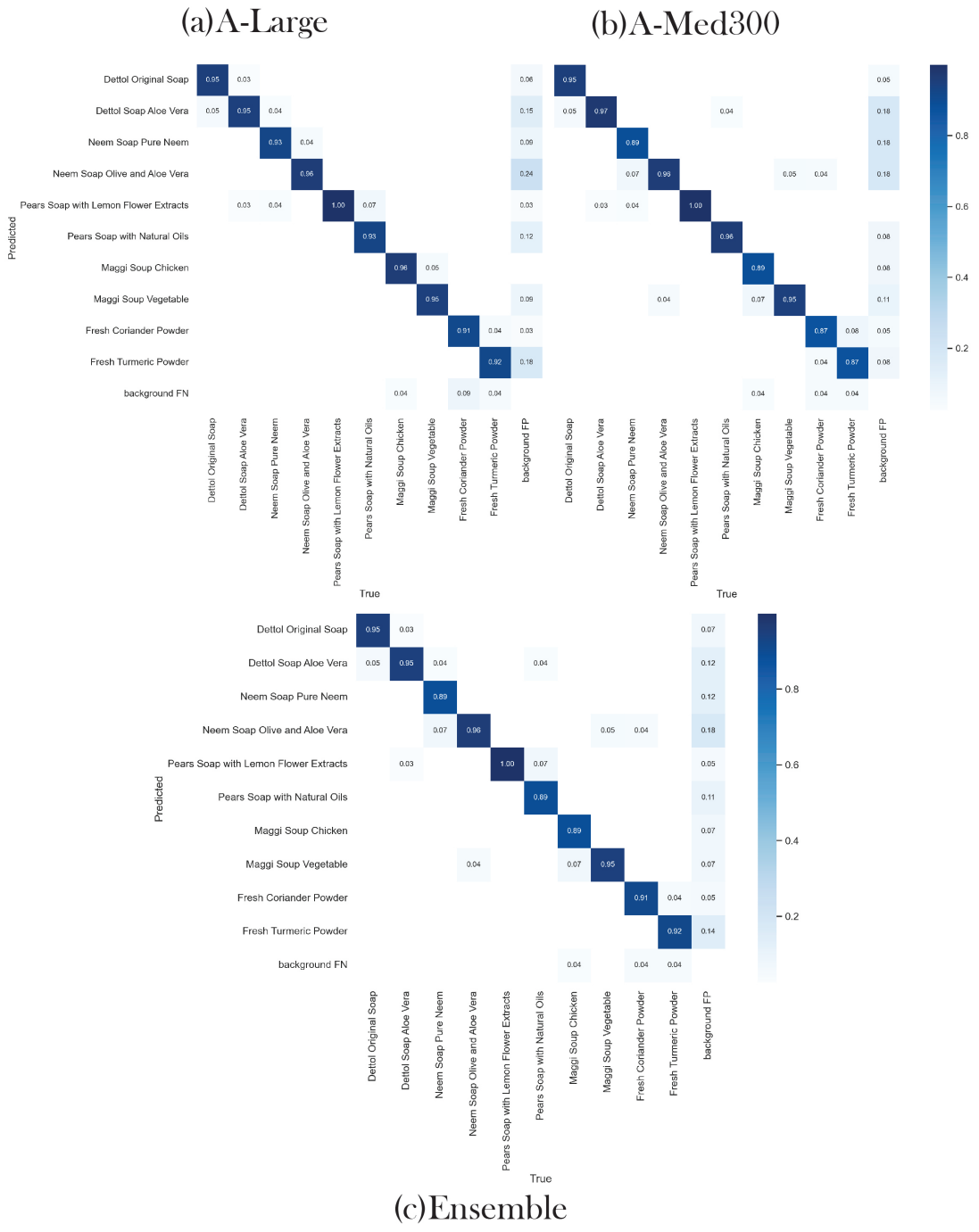


Figure 5.26: Confusion Matrices for Front labels of Test Set - 4 (a) Aruco_Large (b) Aruco_Medium_300 (c)Ensemble_Aruco

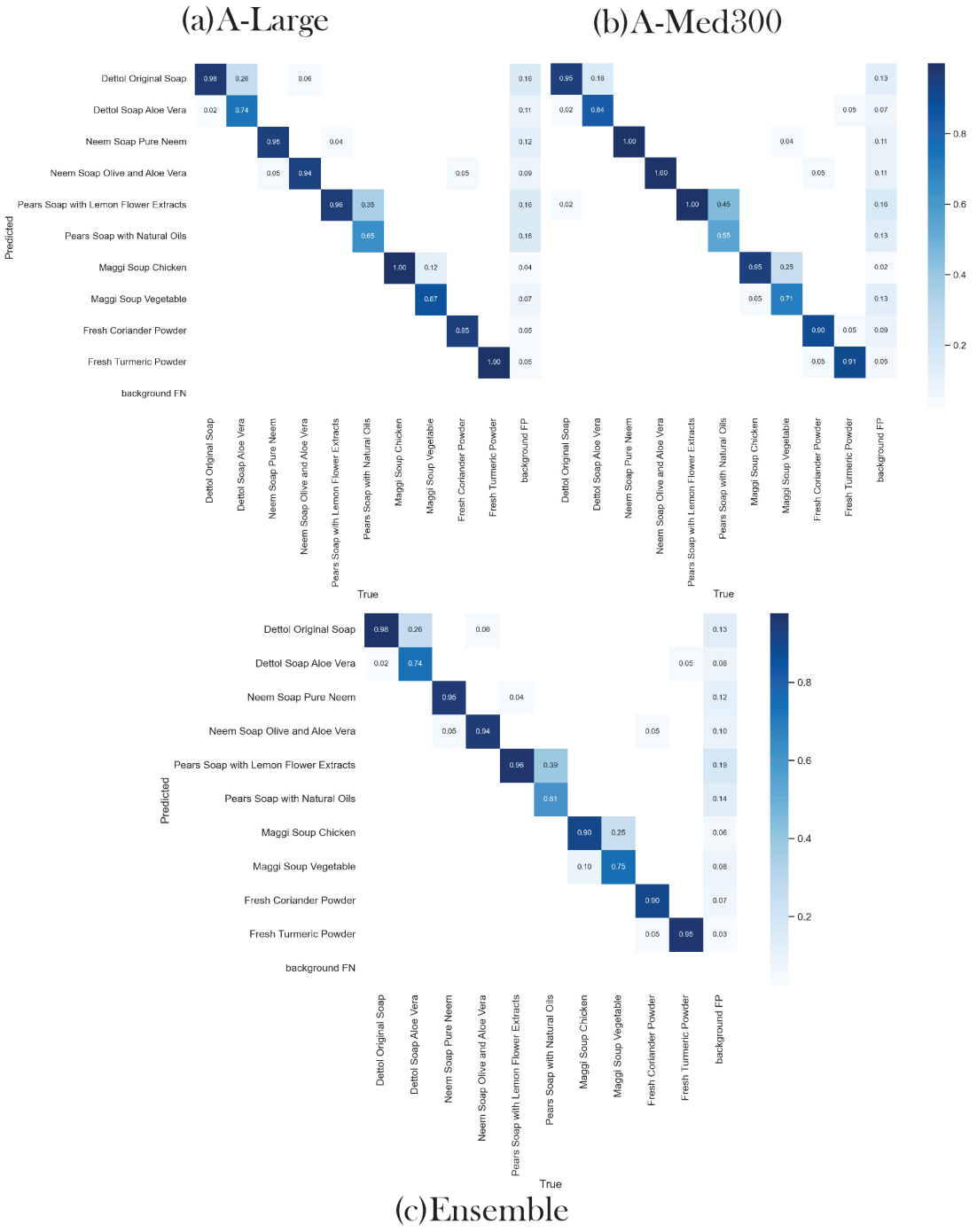


Figure 5.27: Confusion Matrices for Back labels of Test Set - 4 (a) Aruco_Large (b) Aruco_Medium_300 (c)Ensemble_Aruco

In Figure , we notice that the Medium_300 model shows lesser confusion between similar pairs but the Larger model shows lesser confusion between non similar products. The Ensemble model tries to do the best of both worlds, it is more

apparant from the mAP values in Figure 5.28. But the models perform mostly within similar range of accuracy.

In Figure , we observe a bias towards some products, for example Dettol Original and Pears Lemon Extract. This is due to slight unbalance in dataset. The confusion between similar pairs here is very significant as in case of most products the back label doesn't have too much distinguishable details from its variants. This is most apparent in the Pears classes as the difference in the back labels are just with the colors of two words, so the confusion is justifiable.

5.12.1 Effect of Different Lightning Conditions

mAP scores for the 3 models are shown in the chart below for both front and back labels in both bright and dark settings.

In the Figure 5.28, each group of 8 bars represent mAP scores for each model. The first subgroup group of 4 bars represent performance on front labels and the other 4 on back labels. First two bars for each subgroup represents mAP@0.5 for bright and dark lighting conditions, the other two represents mAP@0.5:0.95 scores for bright and dark lighting conditions within the given test set slice. All of the models are performing reasonably similar, with the ensemble having a slight edge on both the mAP0.5 and mAP0.5:0.95 matrices.

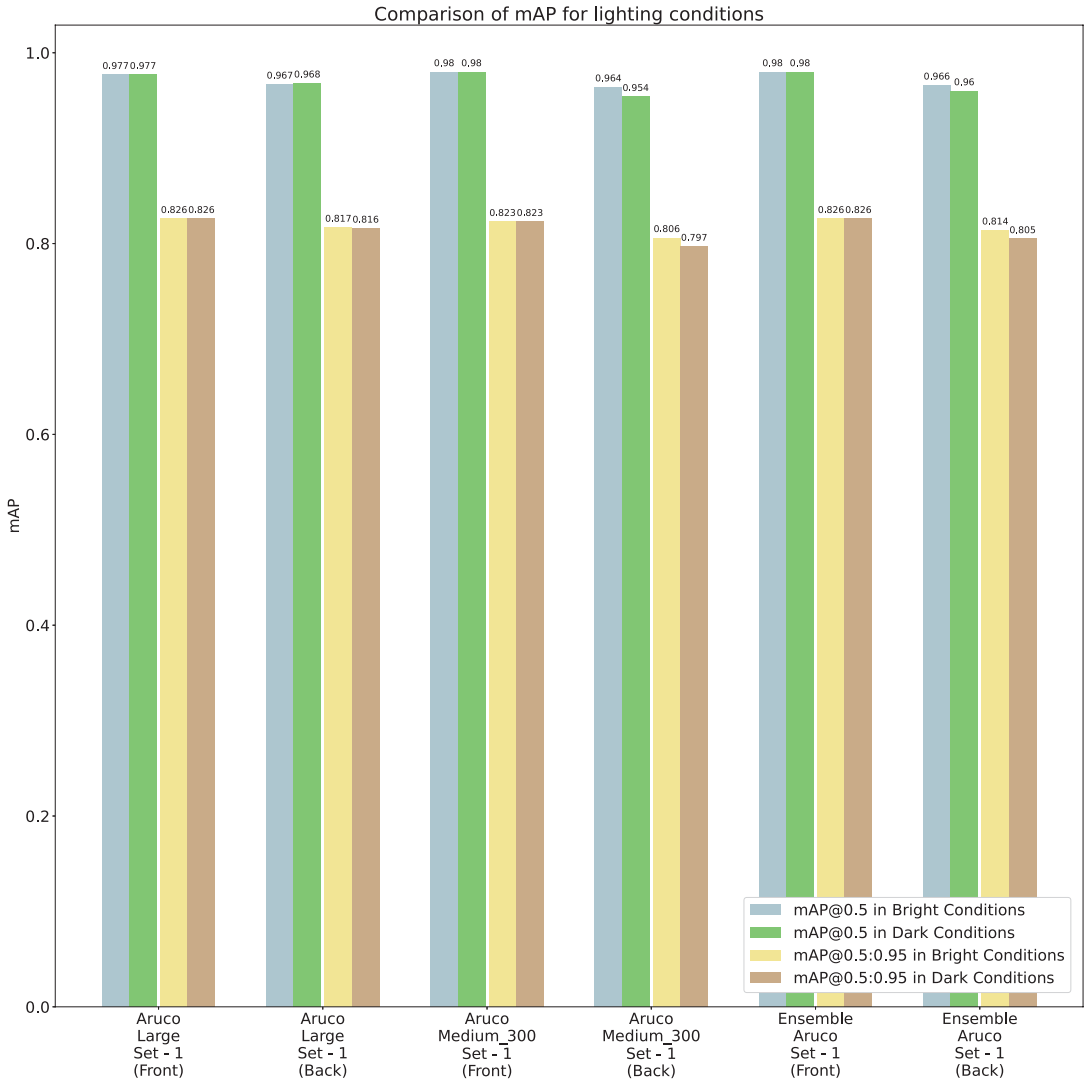


Figure 5.28: Comparison of mAP for different lighting conditions for Similar Looking Products

5.13 Accuracy and Stability Analysis of Markers

5.13.1 Testing Setup

We have tested QR and ArUco markers by sticking them to opposite sides of the same bags. We took nearly identical photos of the bags by just flipping them. So we can assume that both the markers were tested in almost identical scenario.

The final test set we used here consisted of 14 images with 30 instances of only weight dependent products.

Figure 5.29: Sample of Marker Analysis Test Set

5.13.2 Comparison of Accuracy

In this section, we compare QR and Hybrid ArUco Markers accuracy on the test set mentioned above. The bar plot of the results are shown below:

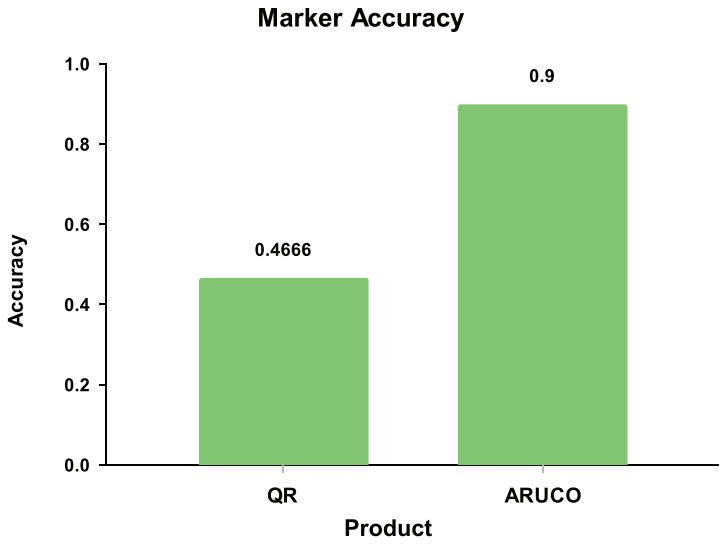


Figure 5.30: Comparison of Accuracy between QR code and Hybrid ArUco Markers

As we can see from the Figure 5.30, ArUco Markers perform much better without extra processing as they are much simpler from QR codes. The simplicity of ArUco

markers help them to be still recognizable if the image is a bit blurred or distorted. But QR codes being complex are more prone to be affected by distortion and blur.

5.13.3 Comparison of Detection Stability

In this section, to test accuracy with varying heights we changed the height of the camera gradually and measured the stability of detection on a single product. Here we defined stability as the ratio of total frames scanned and the total number of frames the product was correctly recognized in.

$$Stability = TotalCorrectRecognition/TotalFrames$$

The results are show in the graph below:

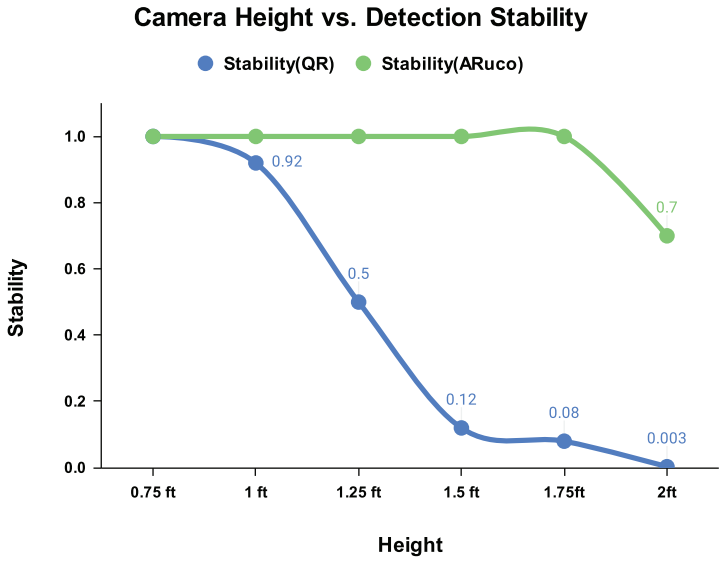


Figure 5.31: Detection Stability vs Height Comparison between QR code and Hybrid ArUco Markers

We can see from the Figure 5.33 that ArUco markers are quite stable through out a range of distances. In case of QR code detection becomes quite shaky above 1 ft height for our print size of 3.5cmx3.5cm, while the Hybrid ArUco Marker performs much better up to 1.75ft. This is why ArUco markers are a better choice for detection from longer distances. However, The distance of detection can be increased more for both markers if we use larger prints.

5.14 Analysis of in Terms of Speed

5.14.1 Test Setup

We tried to simulate a conveyor belt scenario by using a treadmill (OMA Starlet52). The products were bunched up and placed on the treadmill, then we set the speed of the treadmill at different values and took videos of the products while they were in motion. We annotated the several frames from each video and created multiple test sets based on them. It is worth noting that there was a lot of reflection on the packing due to the position of the treadmill in the house. And each product was recorded not more than 2 times per speed, so the a small fluctuation in the data may cause the results to oscillate quite a bit.

Figure 5.32: Treadmill used to carry out the test

Figure 5.33: Frame sample of test set

5.14.2 mAP vs Speed Analysis

In this section, we analyze the data we found after running the test. mAP vs Speed curves for most of the classes are shown below:

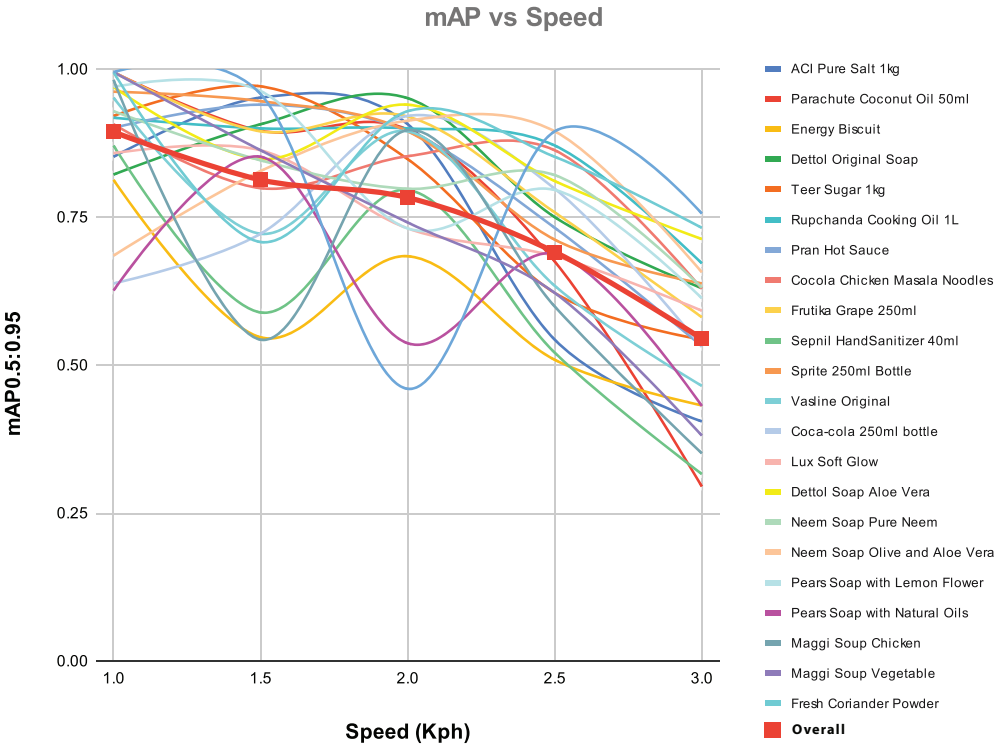


Figure 5.34: mAP vs Speed Comparison for Front Labels

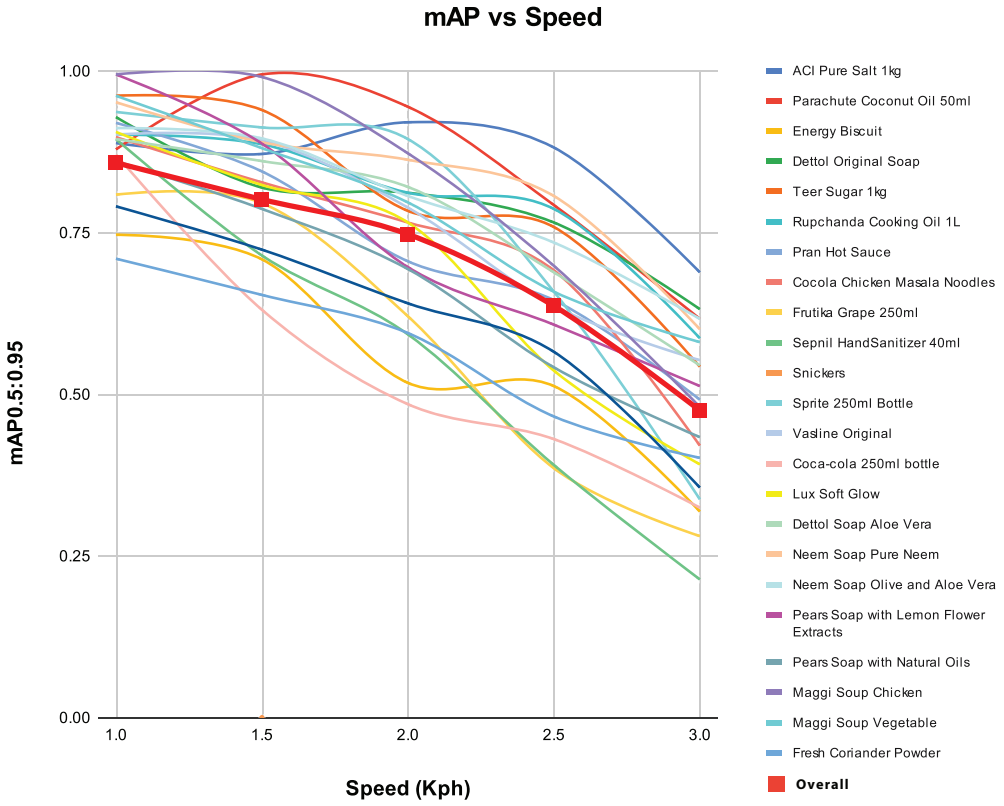


Figure 5.35: mAP vs Speed Comparison for Back Labels

In the Figure 5.34, we can notice that there are several fluctuations in the data due to insufficient instances but we can observe the overall mAP curve falling down gradually with increasing speed. A lot of the mAP values for individual classes are falling drastically, so it can be said that if we took data for higher speed we would see a similar effect in the overall curve.

Also in Figure 5.35, we see a similar trend. Here the slope of the fall is greater, as back labels of products don't usually have that much distinguishable features to begin with and with increasing speed it becomes even more harder to recognize them, From the results we can conclude that for effective detection we should keep the speed of the conveyor belt well below 2 kph.

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

A CV-based billing system for retail products has been discussed in this project. The objective of the project was to replace traditional barcode scanners in supermarkets to speed up the checkout process. We have implemented a GUI-based checkout application that works on a YOLO-based object detection system to bill the products. We also analyze the efficiency of the YOLO-based models and how several factors like data augmentation and the increase of densely arranged images affect the performance. For detecting weight-dependent products, we have implemented ArUco Markers in the packaging of these products. The performance evaluation of ArUco markers turned out to be better than QR codes in many aspects like distance from the camera, lighting conditions, image resolution.

6.1 Discussion

In Chapter 5, we analyzed our models on multiple test sets each focusing on different aspects of object detection. We have found that adding a few densely arranged product photos substantially increased our performance. It is apparent from the data augmentation also adds a significant improvement to the models. In the case of hyperparameter tuning, we observed some improvement in the quality of real-time detection but showed a dip in performance on the other test sets. This is due to the fact that we only could train 150 generations and that to up to 10 epochs which isn't sufficient to generate a hyperparameter-based performance

improvement in the models. But we observed that creating an ensemble with a large model and medium model yielded the best results. The more robustly trained medium model is mostly able to patch out the scenarios where the larger model overfits due to its more complex nature.

In the several tests we conducted, the final model (Ensemble_Aruco) performed reasonably well in both front and back-facing labels, but the improvement over its component models were not overly significant in most cases. The model also performs well independent of lighting conditions, only a small dip in performance is observed in darker environments. Back labels in the darker lighting conditions proved to be the most difficult for the models, but our final model performs reasonably well with mAP@0.5 and mAP@0.5:0.95 around 0.96 and 0.827.

Initially, we trained our model for detecting QR codes on the packaging of weight-dependent products like rice, beef, potato. The detection performance was not optimal because QR codes could not be detected from large distances. After all, the code contains too many details of bit patterns that get lost or blurry with increasing distances from the camera. Therefore, we sought to implement a more suitable marker for our project, which can perform better than QR codes. Based on the findings of this research, [35] which concluded that ArUco markers perform better than QR codes in many aspects, we decided to integrate ArUco Markers in the packaging of weight-dependent products. Overall, implementing this fiducial marker has facilitated our project in achieving a reasonable detection accuracy even from long distances, which could not be detected optimally earlier with QR codes. Conventionally, ArUco Markers are used in augmented reality applications. Therefore, incorporating this marker in supermarket product recognition and combining two different classes of ArUco markers in order to overcome its limitations is a unique approach proposed by our project.

We have taken video feeds of batches of grocery products running over a treadmill at 60 fps, 1080p with the camera of a Pocophone F1 phone. The treadmill has been used as a prototype of a checkout conveyor belt mechanism. Since the lowest speed of the treadmill was 1 km/hr, this approach was not able to properly replicate the real-life scenario, where the conveyor speed is typically lower. We have also taken video feeds at 30 fps, 480p but the detection performance using these videos is

not well enough compared to the detection performance using videos taken at 60 fps, 1080p. For evaluating the detection accuracy, we have regulated batches of grocery products over the treadmill at 1kph, 1.5kph, 2kph, 2.5kph, and 3kph. Our model performs well in lower speed limits of the treadmill.

6.2 Future Scopes

Apart from the results stated in our analysis, there are certain limitations to our system that can be improved in the future by applying different methods. The limitations and scopes for improvement are as follows:

1. As our approach towards detecting products is based on the shape and pattern of the labels, the model will fail to detect differences between products that have exactly similar-looking packaging but have different sizes. For example, Coca-cola bottles of 1L and 2L may not be detected as different products by our system. This may be solved by adding extra conditions for bounding box sizes and product orientation.

2. Single products cannot be removed from the list, this can be solved by improving the UI.

3. Even though we have created a CV-based solution, the system still requires a cashier. We can modify the system, incorporating a conveyor belt mechanism similar to the ARC Vision System [14] and develop an automated solution that will have all the advantages of our system.

4. Currently, we place a batch of retail products under the camera for detection, manually lock these items in the billing UI frame and then replace those with a new batch of products for further locking in the bill section. In the real-life scenario, the conveyor belt will carry products at a constant speed. So, we need to implement a system that is able to automatically insert continuously arriving products in the bill section without a lock mechanism. An alternative approach can be to regulate the conveyor belt to stop under the camera for a batch of products to be detected and locked for billing and

then to start moving the belt to stop again until a new batch of products arrives under the camera. Further research is needed to determine which approach will perform better.

5. We can improve our current Hybrid ArUco Markers by adding more classes of markers in the black boxes. This approach will facilitate to add more categories of weight-dependent products in the inventory and improve the detection performance. If we have to deal with a lot more than 100 weight dependent products then there maybe more confusion between the two classes of ArUco markers and so an additional orientation detection step may need to be implemented. This can also be done by placing special markers on the empty black boxes.

6. Even though real-time detection performance may not be absolutely necessary for billing purposes as it is not essential to analyze every frame in a supermarket scenario. YOLO-based real-time detection works well on GPU-based systems which are expensive to deploy. More research is needed to optimize the model for CPU-based detection in order to make it cost-effective. One solution can be using ONNX [108] to convert to a more CPU-optimized framework like OpenVINO [109] will substantially improve performance on CPU-based systems. The inclusion of automated product type-based sorting and packaging can be a good future scope for the system, in order to move towards a fully automated supermarket experience.

BIBLIOGRAPHY

[1] S. J. Pan and Q. Yang, "A survey on transfer learning," IEEE Transactions on Knowledge and Data Engineering, vol. 22, no. 10, pp. 1345–1359, 2010.

[2] Z. Zou, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," 2019.

[3] M. Rieder and R. Verbeet, "Robot-human-learning for robotic picking processes," in Hamburg International Conference of Logistics (HICL) 2019. epubli GmbH, 2019, pp. 87–114.

[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Redmon_You_Only_Look_CVPR_2016_paper.html

[5] M. Menegaz. Understanding yolo. [Online]. Available: https://medium.com/hackernoon/understanding-yolo-f5a74bbc7967

[6] Cnn bounding box predictions. [Online]. Available: http://datahacker.rs/deep-learning-bounding-boxes/

[7] L. Weng. Object detection part 4: Fast detection models. [Online]. Available: https://lilianweng.github.io/lil-log/2018/12/27/object-detection-part-4.html

[8] R. Takahashi, T. Matsubara, and K. Uehara, "Data augmentation using random image cropping and patching for deep cnns," IEEE Transactions on Circuits and Systems for Video Technology, vol. 30, no. 9, pp. 2917–2931, 2019.

[9] S. I. Nikolenko et al., "Synthetic data for deep learning," arXiv preprint arXiv:1909.11512, vol. 3, p. 11, 2019.

[10] Detection of ArUco Markers, Last accessed on October 2021. [Online]. Available: https://docs.opencv.org/3.4/d5/dae/tutorial_aruco_detection.html

[11] Laughing-q. Yolov5 network structure. [Online]. Available: https://blog.csdn.net/Q1u1NG/article/details/107511465

[12] R. Morabito and F. C. R. de Lima, "A markovian queueing model for the analysis of user waiting times in supermarket checkouts," International Journal of Operations and Quantitative Management, vol. 10, no. 2, pp. 165–177, July 2004.

[13] Amazon Dash Cart, Last accessed on October 2021. [Online]. Available: https://www.amazon.com/b?ie=UTF8&node=21289116011

[14] S. T. Bukhari, A. W. Amin, M. A. Naveed, and M. R. Abbas, "Arc: A vision-based automatic retail checkout system," 2021.

[15] M. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, and G. Hinton, "On rectified linear units for speech processing," in 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, 2013, pp. 3517–3521.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in Proceedings of the IEEE International Conference on Computer Vision (ICCV), December 2015.

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," The journal of machine learning research, vol. 15, no. 1, pp. 1929–1958, 2014.

[18] K. Georgiadis, F. Kalaganis, P. Migkotzidis, E. Chatzilari, S. Nikolopoulos, e. D. Kompatsiaris, Ioannis", D. Giakoumis, M. Vincze, and A. Argyros, "A computer vision system supporting blind people - the supermarket case," in

Computer Vision Systems. Cham: Springer International Publishing, 2019, pp. 305–315.

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1409.1556

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016.

[21] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," ser. AAAI'17. AAAI Press, 2017, p. 4278–4284.

[22] T. Winlock, E. Christiansen, and S. Belongie, "Toward real-time grocery detection for the visually impaired," in 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops, 2010, pp. 49–56.

[23] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2, ser. IJCAI'81. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1981, p. 674–679.

[24] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," Computer Vision and Image Understanding, vol. 110, no. 3, pp. 346–359, 2008, similarity Matching in Computer Vision and Multimedia. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314207001555

[25] L. Barrington, T. K. Marks, J. Hui-wen Hsiao, and G. W. Cottrell, "Nimble: A kernel density model of saccade-based visual memory," Journal of Vision, vol. 8, no. 14, pp. 17–17, 11 2008. [Online]. Available: https://doi.org/10.1167/8.14.17

[26] W. Geng, F. Han, J. Lin, L. Zhu, J. Bai, S. Wang, L. He, Q. Xiao, and Z. Lai, "Fine-grained grocery product recognition by one-shot learning," in Proceedings of the 26th ACM International Conference on Multimedia, ser. MM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1706–1714. [Online]. Available: https://doi.org/10.1145/3240508.3240522

[27] J. Liu and Y. Liu, "Grasp recurring patterns from a single view," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2013.

[28] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," Commun. ACM, vol. 24, no. 6, p. 381–395, Jun. 1981. [Online]. Available: https://doi.org/10.1145/358669.358692

[29] A. Franco, D. Maltoni, and S. Papi, "Grocery product detection and recognition," Expert Systems with Applications, vol. 81, pp. 163–176, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417417301227

[30] M. Merler, C. Galleguillos, and S. Belongie, "Recognizing groceries in situ using in vitro training data," in 2007 IEEE Conference on Computer Vision and Pattern Recognition, 2007, pp. 1–8.

[31] A. Tonioni and L. Di Stefano, "Product recognition in store shelves as a sub-graph isomorphism problem," in Image Analysis and Processing - ICIAP 2017, S. Battiato, G. Gallo, R. Schettini, and F. Stanco, Eds. Cham: Springer International Publishing, 2017, pp. 682–693.

[32] A. Abdel-Hakim and A. Farag, "Csift: A sift descriptor with color invariant characteristics," in 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), vol. 2, 2006, pp. 1978–1983.

[33] K. van de Sande, T. Gevers, and C. Snoek, "Evaluating color descriptors for object and scene recognition," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 9, pp. 1582–1596, 2010.

[34] S. Leutenegger, M. Chli, and R. Y. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in 2011 International Conference on Computer Vision, 2011, pp. 2548–2555.

[35] M. Elgendy, T. Guzsvinecz, and C. Sik-Lanyi, "Identification of markers in challenging conditions for people with visual impairment using convolutional neural network," Applied Sciences, vol. 9, no. 23, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/23/5110

[36] C930E BUSINESS WEBCAM DATASHEET, Last accessed on October 2021. [Online]. Available: https://www.logitech.com/content/dam/logitech/en_us/video-collaboration/pdf/c930e-datasheet.pdf

[37] H.-I. Suk, "Chapter 1 - an introduction to neural networks and deep learning," in Deep Learning for Medical Image Analysis, S. K. Zhou, H. Greenspan, and D. Shen, Eds. Academic Press, 2017, pp. 3–24. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B978012810408800002X

[38] S. Agatonovic-Kustrin and R. Beresford, "Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research," Journal of Pharmaceutical and Biomedical Analysis, vol. 22, no. 5, pp. 717–727, 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0731708599002721

[39] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," Psychological Review, vol. 65, no. 6, p. 386–408, 1958. [Online]. Available: https://psycnet.apa.org/doi/10.1037/h0042519

[40] P. Y. Simard, Y. A. LeCun, J. S. Denker, and B. Victorri, Transformation Invariance in Pattern Recognition — Tangent Distance and Tangent Propagation. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 239–274.

[41] L. Torrey and J. Shavlik, Transfer Learning. IGI Global, 2010, pp. 242–264.

[42] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in Proceedings of

the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2014.

[43] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," Commun. ACM, vol. 60, no. 6, p. 84–90, May 2017. [Online]. Available: https://doi.org/10.1145/3065386

[44] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection," in Advances in Neural Information Processing Systems, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., vol. 26. Curran Associates, Inc., 2013. [Online]. Available: https://proceedings.neurips.cc/paper/2013/file/f7cade80b7cc92b991cf4d2806d6bd78-Paper.pdf

[45] A Gentle Introduction to Object Recognition With Deep Learning, Last accessed on October 2021. [Online]. Available: https://machinelearningmastery.com/object-recognition-with-deep-learning/

[46] R. Girshick, "Fast r-cnn," in Proceedings of the IEEE international conference on computer vision, 2015, pp. 1440–1448.

[47] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask r-cnn," in Proceedings of the IEEE International Conference on Computer Vision (ICCV), Oct 2017.

[48] R-CNN – Neural Network for Object Detection and Semantic Segmentation, Last accessed on October 2021. [Online]. Available: https://neurohive.io/en/popular-networks/r-cnn/

[49] C. Chen, M.-Y. Liu, O. Tuzel, and J. Xiao, "R-cnn for small object detection," in Computer Vision – ACCV 2016, S.-H. Lai, V. Lepetit, K. Nishino, and Y. Sato, Eds. Cham: Springer International Publishing, 2017, pp. 214–230.

[50] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), vol. 2, 2006, pp. 2169–2178.

[51] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 37, no. 9, pp. 1904–1916, 2015.

[52] A. Bochkovskiy, C.-Y. Wang, and H.-y. Liao, "Yolov4: Optimal speed and accuracy of object detection," 04 2020.

[53] G. Maindola. A brief history of yolo object detection models from yolov1 to yolov5. [Online]. Available: https://machinelearningknowledge.ai/a-brief-history-of-yolo-object-detection-models/#YOLOv4_8211_Optimal_Speed_and_Accuracy_of_Object_Detection

[54] Yolo: Real-time object detection. [Online]. Available: https://pjreddie.com/darknet/yolo/

[55] R. Gandhi. R-cnn, fast r-cnn, faster r-cnn, yolo — object detection algorithms. [Online]. Available: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[56] Overview of the yolo object detection algorithm. [Online]. Available: https://medium.com/@ODSC/overview-of-the-yolo-object-detection-algorithm-7b52a745d3e0

[57] J. Deng, X. Xuan, W. Wang, Z. Li, H. Yao, and Z. Wang, "A review of research on object detection based on deep learning," Journal of Physics: Conference Series, vol. 1684, p. 012028, nov 2020. [Online]. Available: https://doi.org/10.1088/1742-6596/1684/1/012028

[58] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016.

[59] A. V. Thatte. Evolution of yolo — yolo version 1. [Online]. Available: https://towardsdatascience.com/evolution-of-yolo-yolo-version-1-afb8af302bd2

[60] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in Proceedings of the 32nd International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37.

Lille, France: PMLR, 07–09 Jul 2015, pp. 448–456. [Online]. Available: https://proceedings.mlr.press/v37/ioffe15.html

[61] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017.

[62] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," International Journal of Computer Vision, vol. 115, 09 2014.

[63] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016.

[64] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018.

[65] C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, "Cspnet: A new backbone that can enhance learning capability of cnn," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, June 2020.

[66] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017.

[67] J. Solawetz. Yolov5 new version - improvements and evaluation. [Online]. Available: https://blog.roboflow.com/yolov5-improvements-and-evaluation/

[68] ——. How to train yolov5 on a custom dataset. [Online]. Available: https://blog.roboflow.com/how-to-train-yolov5-on-a-custom-dataset/

[69] A. Mikołajczyk and M. Grochowski, "Data augmentation for improving deep learning in image classification problem," in 2018 International Interdisciplinary PhD Workshop (IIPhDW), 2018, pp. 117–122.

[70] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," Journal of Big Data, vol. 6, no. 1, pp. 1–48, 2019.

[71] I. Limit. Composing images with python for synthetic datasets. [Online]. Available: https://www.immersivelimit.com/tutorials/composing-images-with-python-for-synthetic-datasets

[72] S. Pokhrel. Image data labelling and annotation-everything you need to know. [Online]. Available: https://towardsdatascience.com/image-data-labelling-and-annotation-everything-you-need-to-know-86ede6c684b1

[73] D. Choi, C. J. Shallue, Z. Nado, J. Lee, C. J. Maddison, and G. E. Dahl, "On empirical comparisons of optimizers for deep learning," CoRR, vol. abs/1910.05446, 2019. [Online]. Available: http://arxiv.org/abs/1910.05446

[74] N. S. Keskar and R. Socher, "Improving generalization performance by switching from adam to SGD," CoRR, vol. abs/1712.07628, 2017. [Online]. Available: http://arxiv.org/abs/1712.07628

[75] N. Ketkar, "Stochastic gradient descent," in Deep learning with Python. Springer, 2017, pp. 113–132.

[76] P. Probst, M. N. Wright, and A.-L. Boulesteix, "Hyperparameters and tuning strategies for random forest," WIREs Data Mining and Knowledge Discovery, vol. 9, no. 3, p. e1301, 2019. [Online]. Available: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1301

[77] J. Brownlee. What is the difference between a parameter and a hyperparameter? [Online]. Available: https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/

[78] G. Jocher. Hyperparameter evolution. [Online]. Available: https://github.com/ultralytics/yolov5/issues/607

[79] J. H. Holland, "Genetic algorithms," Scientific American, vol. 267, no. 1, pp. 66–73, 1992. [Online]. Available: http://www.jstor.org/stable/24939139

[80] L. Tani, D. Rand, C. Veelken, and M. Kadastik, "Evolutionary algorithms for hyperparameter optimization in machine learning for application in high energy physics," 11 2020.

[81] QR Code Basic, Last accessed on October 2021. [Online]. Available: https://www.qr-code-generator.com/qr-code-marketing/qr-codes-basics/

[82] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and M. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," Pattern Recognition, vol. 47, p. 2280–2292, 06 2014.

[83] S. Suzuki and K. be, "Topological structural analysis of digitized binary images by border following," Computer Vision, Graphics, and Image Processing, vol. 30, no. 1, pp. 32–46, 1985. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0734189X85900167

[84] D. H. DOUGLAS and T. K. PEUCKER, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," Cartographica: The International Journal for Geographic Information and Geovisualization, vol. 10, no. 2, pp. 112–122, 1973. [Online]. Available: https://doi.org/10.3138/FM57-6770-U75U-7727

[85] N. Otsu, "A threshold selection method from gray-level histograms," IEEE Transactions on Systems, Man, and Cybernetics, vol. 9, no. 1, pp. 62–66, 1979.

[86] Daraz, Last accessed on October 2021. [Online]. Available: https://www.daraz.com.bd/

[87] The Selenium Browser Automation Project, Last accessed on October 2021. [Online]. Available: https://www.selenium.dev/documentation//

[88] Beautiful Soup Documentation, Last accessed on October 2021. [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[89] dupeGuru, Last accessed on October 2021. [Online]. Available: https://dupeguru.voltaicideas.net/

[90] Labelimg, Last accessed on October 2021. [Online]. Available: https://github.com/tzutalin/labelImg

[91] R. Khandelwal. Coco and pascal voc data format for object detection. [Online]. Available: https://towardsdatascience.com/coco-data-format-for-object-detection-a4c5eaf518c5

[92] A. Khan. Convert pascalvoc annotations to yolo. [Online]. Available: https://gist.github.com/Amir22010/a99f18ca19112bc7db0872a36a03a1ec

[93] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in Advances in Neural Information Processing Systems, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

[94] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in Computer Vision – ECCV 2014, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 740–755.

[95] G. Jocher, A. Stoken, J. Borovec, NanoCode012, A. Chaurasia, TaoXie, L. Changyu, A. V, Laughing, tkianai, yxNONG, A. Hogan, lorenzomammana, AlexWang1900, J. Hajek, L. Diaconu, Marc, Y. Kwon, oleg, wanghaoyang0106, Y. Defretin, A. Lohia, ml5ah, B. Milanko, B. Fineran, D. Khromov, D. Yiwei, Doug, Durgesh, and F. Ingham, "ultralytics/yolov5: v5.0 - YOLOv5-P6 1280 models, AWS, Supervise.ly and YouTube integrations," Apr. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.4679653

[96] Z.-H. Zhou, J. Wu, and W. Tang, "Ensembling neural networks: Many could be better than all," Artificial Intelligence, vol. 137, no. 1, pp. 239–263, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S000437020200190X

[97] L. Hansen and P. Salamon, "Neural network ensembles," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 12, no. 10, pp. 993–1001, 1990.

[98] Openbase, pyzbar: Documentation, Last accessed on February 2022. [Online]. Available: https://openbase.com/python/pyzbar/documentation

[99] S. Nayak, "Augmented reality using aruco markers in opencv (c++ / python): Learnopencv ," May 2021. [Online]. Available: https://learnopencv.com/augmented-reality-using-aruco-markers-in-opencv-c-python/

[100] W. Galitz, The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques, ser. Wiley Desktop Editions. Wiley, 2007. [Online]. Available: https://books.google.com.bd/books?id=Q3Xp_Awu49sC

[101] J. Willman, Overview of PyQt5. Berkeley, CA: Apress, 2021, pp. 1–42. [Online]. Available: https://doi.org/10.1007/978-1-4842-6603-8_1

[102] seaborn.pairplot, Last accessed on October 2021. [Online]. Available: https://seaborn.pydata.org/generated/seaborn.pairplot.html

[103] R. Padilla, S. L. Netto, and E. A. B. da Silva, "A survey on performance metrics for object-detection algorithms," in 2020 International Conference on Systems, Signals and Image Processing (IWSSIP), 2020, pp. 237–242.

[104] J. Brownlee. https://machinelearningmastery.com/confusion-matrix-machine-learning/. [Online]. Available: https://machinelearningmastery.com/confusion-matrix-machine-learning/

[105] A. Rahmani, "Adapting google translate for english-persian cross-lingual information retrieval in medical domain," in 2017 Artificial Intelligence and Signal Processing Conference (AISP), 2017, pp. 43–46.

[106] COCO Dectection Evaluation, Last accessed on October 2021. [Online]. Available: https://cocodataset.org/#detection-eval

[107] Open Images Challenge 2018 - object detection track - evaluation metric, Last accessed on October 2021. [Online]. Available: https://storage.googleapis.com/openimages/web/object_detection_metric.html

[108] T. Le, G.-T. Bercea, T. Chen, A. Eichenberger, H. Imai, T. Jin, K. Kawachiya, Y. Negishi, and K. O'Brien, "Compiling onnx neural network models using mlir," 08 2020.

[109] Y. Gorbachev, M. Fedorov, I. Slavutin, A. Tugarev, M. Fatekhov, and Y. Tarkan, "Openvino deep learning workbench: Comprehensive analysis and tuning of neural networks inference," in Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops, Oct 2019.

APPENDIX A: LIST OF ACRONYMS

COCO	Common Objects In Context
CNN	Convolutional Neural Network
CSPNet	Cross Stage Partial Network
CV	Computer Vision
DNN	Deep Neural Network
FPN	Feature Pyramid Network
GA	Genetic Algorithms
GAN	Generative Adversarial Networks
GUI	Graphical User Interface
FOV	Field of vision
OpenCV	Open Source Computer Vision Library
PANet	Path Aggregation Network
RANSAC	Random Sample Consensus
ReLu	Rectified Linear Unit
ROI	Region of Interest
SIFT	Scale Invariant Feature Transform
SGD	Stochastic Gradient Descent
SVM	Support Vector Machines
YOLO	You Only Look Once
IoU	Intersection over Union