

**SMART CASHIERLESS CHECKOUT  
SYSTEM FOR RETAIL USING MACHINE  
VISION**

**LEE REN YI**

**UNIVERSITI TUNKU ABDUL RAHMAN**

**SMART CASHIERLESS CHECKOUT SYSTEM FOR RETAIL USING  
MACHINE VISION**

**LEE REN YI**

**A project report submitted in partial fulfilment of the  
requirements for the award of Bachelor of Engineering  
(Honours) Mechatronics Engineering**

**Lee Kong Chian Faculty of Engineering and Science  
Universiti Tunku Abdul Rahman**

**May 2022**

**DECLARATION**

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature :   
\_\_\_\_\_

Name : Lee Ren Yi  
\_\_\_\_\_

ID No. : 17UEB02805  
\_\_\_\_\_

Date : 25 April 2022  
\_\_\_\_\_

**APPROVAL FOR SUBMISSION**

I certify that this project report entitled “**SMART CASHIERLESS CHECKOUT SYSTEM FOR MACHINE VISION**” was prepared by **LEE REN YI** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) Mechatronics Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature

:



Supervisor

:

Dr. Chai Tong Yuen

Date

:

24 April 2022

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of Universiti Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2022, Lee Ren Yi. All right reserved.

## ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Dr. Chai Tong Yuen for his invaluable advice, guidance and his enormous patience throughout the development of the research.

In addition, I would also like to express my gratitude to families, friends and UTAR lecturers for their continuous mental support and encouragement that served as motivation in completing this project.

## ABSTRACT

As Corona Virus 2019 (COVID-19) pandemic strikes the world, retail industry has been severely impacted especially in its daily operation due to the restriction of workforce and most of the face-to-face services, including checkout, are associated with high risk of developing spread chain of COVID-19 virus. Despite there are multiple computer vision-based solutions available in the field such as on-shelf checkout and sensor fusion, but they can be expensive and may require overhaul of stores, which is unfeasible for small retail stores. Therefore, a software prototype of intelligent cashierless checkout system is proposed to help small-scale retail stores in minimizing the risk of developing COVID-19 virus spread chain as well as the workforce requirement during checkout using state-of-the-art object detection models. This project was performed in 2 parts where the first stage involved an image synthesis algorithm to automatically produce visually realistic product images using Generative Adversarial Network (GAN). Several GAN architectures such as CycleGAN and AttentionGAN were studied and compared in terms of their effectiveness in generating realistic shadow in actual checkout scenario. CycleGAN results in more realistic shadow with Fréchet inception distance (FID) of 40.99. In the following stage, a publicly available dataset, MVTec D2S dataset were used to benchmark multiple object detection models used for product recognition. By using You Only Look Once (YOLO) v5L as the baseline model, several improved models were developed by replacing the backbone structure with other light-weight architectures to improve computation efficiency when deployed on edge devices. After training the model with dataset generated in previous stage, the proposed model with MobileNet V3 surpassed baseline model in terms of inference time, with only 0.142s while maintaining high Mean Average Precision (mAP) of 98.2% and Checkout Accuracy (cAcc) of 89.17% on Jetson Nano.

## TABLE OF CONTENTS

<b>DECLARATION</b>		<b>i</b>
<b>APPROVAL FOR SUBMISSION</b>		<b>ii</b>
<b>ACKNOWLEDGEMENTS</b>		<b>iv</b>
<b>ABSTRACT</b>		<b>v</b>
<b>TABLE OF CONTENTS</b>		<b>vi</b>
<b>LIST OF TABLES</b>		<b>x</b>
<b>LIST OF FIGURES</b>		<b>xi</b>
<b>LIST OF SYMBOLS / ABBREVIATIONS</b>		<b>xiii</b>
<b>LIST OF APPENDICES</b>		<b>xvii</b>
 <b>CHAPTER</b>		
<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
1.1	General Introduction	1
1.2	Importance of the Study	2
1.3	Problem Statement	3
1.4	Aim and Objectives	4
1.5	Scope and Limitation of the Study	4
1.6	Contribution of the Study	5
1.7	Outline of the Report	5
 <b>2</b>	 <b>LITERATURE REVIEW</b>	 <b>7</b>
2.1	Existing Cashierless Checkout Approaches	7
2.1.1	Radio Frequency Identification (RFID)	7
2.1.2	Artificial Intelligence (AI) and Computer Vision	8
2.2	Datasets of Retail Products	9
2.2.1	Retail Product Checkout Dataset (RPC)	10
2.2.2	Densely Segmented Supermarket (D2S) Dataset	11



2.2.3	CAPG Grocery Product Dataset (CAPG-GP)	13
2.2.4	Freiburg Groceries	13
2.2.5	Dataset Comparison	15
2.3	Image Synthesis	16
2.3.1	Conventional Data Augmentation	16
2.3.2	Mask based Synthesis	17
2.3.3	Generative Adversarial Network (GAN)	19
2.4	Detection Algorithm	20
2.4.1	Classic Computer Vision Techniques	20
2.4.1.1	SURF + Feature Matching	21
2.4.1.2	Haar Features + AdaBoost & SIFT + SVM	22
2.4.2	Deep Learning based Computer Vision Techniques	23
2.4.2.1	CNN	24
2.4.2.2	State-of-the-Art Models	25
2.4.2.2.1	Faster R-CNN	26
2.4.2.2.2	Mask R-CNN	27
2.4.2.2.3	YOLO	29
2.4.2.2.4	RetinaNet	31
2.4.2.3	Summary	32
2.5	Supportive Elements for Retail Checkout System	33
<b>3</b>	<b>METHODOLOGY AND WORK PLAN</b>	<b>34</b>
3.1	Introduction	34
3.1.1	Overall System Flow	34
3.1.2	Work Plan	35
3.2	Dataset Preparation	36
3.2.1	MVTec D2S Dataset	37
3.2.2	Custom Dataset Preparation	39
3.3	Development of Image Synthesis Framework	41
3.3.1	Binary Mask Extraction	41
3.3.2	Crop and Place Algorithm	42

3.3.3	GAN-based Shadow Synthesis	43
3.3.4	Lighting Variation	46
3.4	Selection of Baseline Object Detection Model	47
3.4.1	State-of-the-Art Model Architectures	48
3.4.2	Preliminary Benchmarking	52
3.5	Model Optimization	53
3.5.1	ShuffleNet V2	53
3.5.2	MobileNet V3	55
3.5.3	GhostNet	56
3.6	Computing Platform	57
3.6.1	TensorRT Acceleration	58
3.7	Evaluation Metrics	60
3.7.1	Fréchet inception distance (FID)	60
3.7.2	Mean Average Precision (mAP)	60
3.7.3	Confusion Matrix	61
3.7.4	Checkout Accuracy (cAcc)	62
3.7.5	Training and Inference Time	63
3.8	Software Development	63
3.8.1	MongoDB Database	63
3.8.2	Tkinter	64
<b>4</b>	<b>RESULTS AND DISCUSSION</b>	<b>65</b>
4.1	Preliminary Benchmarking	65
4.1.1	Quantitative Results	65
4.1.2	Qualitative Results	66
4.2	Effectiveness of Image Synthesis Framework	68
4.2.1	GAN-synthesized Images	68
4.2.2	Effect on Model Performance	70
4.2.2.1	Quantitative Results	71
4.3	Model Improvement with light-weight backbones	72
4.3.1	Quantitative Results	73
4.3.2	Qualitative Analysis	75
4.3.2.1	Performance in Extreme Condition	76
4.3.2.2	Adaptivity to Lighting Variation	77

4.3.3	Training Loss	78
4.4	Software Prototype of Cashierless Checkout System	80
4.4.1	Model Weights Update	80
4.4.2	Price Computation	80
<b>5</b>	<b>CONCLUSIONS AND RECOMMENDATIONS</b>	<b>82</b>
5.1	Conclusions	82
5.2	Recommendations for Future Work	83
	<b>REFERENCES</b>	<b>84</b>
	<b>APPENDICES</b>	<b>94</b>

**LIST OF TABLES**

Table 2.1:	Characteristic Table of Publicly Available Dataset	15
Table 3.1:	Hyperparameters for GAN	43
Table 3.2:	Configuration of Data Augmentation	46
Table 3.3:	Hyperparameters for Product Recognition Model	52
Table 3.4:	Raspberry Pi 4 and Jetson Nano Specifications	58
Table 3.5:	Example of Confusion Matrix	62
Table 4.1:	Quantitative Performance of Representative Models	65
Table 4.2:	Confidence Score of Detected Products	67
Table 4.3:	Shadow Synthesis Results of AttentionGAN and CycleGAN	69
Table 4.4:	Quantitative Results for Different Levels of Image Synthesis	71
Table 4.5:	Quantitative Results for Light-Weight Experimental Models	73
Table 4.7:	Confidence Score of Detected Products in Extreme Condition	76
Table 4.6:	Confidence Score of Detected Products in Low Light	77

## LIST OF FIGURES

Figure 1.1:	Malaysian MCO Activity 2020 (Hirschmann, 2021)	1
Figure 2.1:	RPC Dataset: Training (a) and Validation Images (b)	10
Figure 2.2:	D2S Dataset: Training (a) and Validation Images (b)	12
Figure 2.3:	CAPG-GP Dataset: Training (a) and Test Images (b)	13
Figure 2.4:	Freiburg Dataset: Training (a) and Test Images (b)	14
Figure 2.5:	Mask-Based Image Synthesis	18
Figure 2.6:	Mask extraction and Merging for synthetic data generation (Koturwar, Shiraishi and Iwamoto, 2019)	19
Figure 2.7:	Development of State-of-the-Art model architectures (Boesch, 2021)	26
Figure 3.1:	Proposed Cashierless Checkout System Architecture	34
Figure 3.2:	Waterfall Diagram for Project Prototype Development	36
Figure 3.3:	Samples (a) and Instances Per Class (b) in MVTEC D2S Subset	37
Figure 3.4:	File Structure of YOLO, PASCAL VOC and COCO Dataset	38
Figure 3.5:	Image Acquisition Setup	39
Figure 3.6:	Samples in Raw Dataset	40
Figure 3.7:	VGG Image Annotator	41
Figure 3.8:	Binary Masks of Retail Products	42
Figure 3.9:	Synthesized Training Images	43
Figure 3.10:	Architecture of CycleGAN	44
Figure 3.11:	Architecture of AttentionGAN	45
Figure 3.12:	Samples in Expanded Dataset	47
Figure 3.13:	Instances Per Class in self-prepared dataset	47
Figure 3.14:	YOLOv3 Model Architecture	49

Figure 3.15:	YOLOv5 Architecture	49
Figure 3.16:	Architecture of RetinaNet	51
Figure 3.17:	GPU Specification in Google Colaboratory	52
Figure 3.18:	ShuffleNet V2 Architecture	54
Figure 3.19:	Proposed YOLOv5 with ShuffleNet V2 Backbone	54
Figure 3.20:	Depthwise and Pointwise Convolution Process	55
Figure 3.21:	Hard-Swish Activation Function (Howard et al., 2019)	56
Figure 3.22:	Proposed YOLOv5L with MobileNet V3 Backbone	56
Figure 3.23:	Ghost Module (Han et al., 2020)	57
Figure 3.24:	Proposed YOLOv5 with GhostNet Backbone	57
Figure 3.25:	Jetson Nano (NVIDIA, 2014)	58
Figure 3.26:	TensorRT Quantization (NVIDIA, 2016)	59
Figure 3.27:	MongoDB Admin Portal	63
Figure 4.1:	Ground truth and Predictions of Representative Models	67
Figure 4.2:	Confusion Matrix of Experimental Models	74
Figure 4.3:	Prediction of Experimental Models in Extreme Condition	75
Figure 4.4:	Prediction of Experimental Models in Low Light	77
Figure 4.5:	Loss Curve of YOLOv5 (Baseline)	78
Figure 4.6:	Training and Validation Loss of YOLOv5 (MobileNet V3)	79
Figure 4.7:	Training and Validation Loss of YOLOv5 (GhostNet)	79
Figure 4.8:	Training and Validation Loss of YOLOv5 (ShuffleNet V2)	79
Figure 4.9:	Model Weight Automatic Update Feature	80
Figure 4.10:	Frame Difference and Contour	81
Figure 4.11:	Continuous Update of Shopping Cart	81

## LIST OF SYMBOLS / ABBREVIATIONS

\$	US dollar
%	Percentage
$A$	ROI of current product
$A^b$	Background Attention Mask
$A^f$	Foreground Attention Mask
$B$	ROI of existing products in background image
$C^f$	Foreground Content Mask
$c_{i,p}$	Image intensity at pixel $p$ in image $i$
$G_A$	Attention Mask Generator
$G_C$	Content Mask Generator
G	Gigabytes
$G$	Generator
$GT_{i,k}$	Actual count of $k$ -th product class in $i$ -th image
hrs	hours
s	seconds
M	Megabytes
$N$	Number of images
$P$	Precision
$P_{i,k}$	Predicted count of $k$ -th product class in $i$ -th image
$p_t$	Probability for Focal Loss
$p$	Image pixel
$R$	Recall
T	Terabytes
$\Sigma_r$	Covariance matrix of real images
$\Sigma_g$	Covariance matrix of generated images
$\gamma$	Focusing Factor
$\sigma_p$	Standard deviation of image pixel $p$
$\mu_p$	Average of pixel values at pixel $p$
$\mu_r$	Mean of real images (feature-wise)
$\mu_g$	Mean of fake images (feature-wise)

AdaBoost	Adaptive Boosting
Adam	Adaptive Moment Estimation
AI	Artificial Intelligence
AP	Average Precision
API	Application Programming Interface
ARC	Automatic Retail Checkout
AUC	Area Under the Curve
CA	Checkout Area
cAcc	Checkout Accuracy
CCTV	Closed Circuit Television
CNN	Convolutional Neural Network
COCO	Common Object In Context
COVID-19	Corona Virus 2019
CSI	Camera Serial Interface
CSPNet	Cross Stage Partial Network
CSV	Comma Separated Values
CUDA	Compute Unified Device Architecture
DPNet	Data Priming Network
FCN	Fully Convolutional Network
FID	Frechet Inception Distance
FLOPs	Floating-Point Operations
FLOPS	Floating-Point Operations Per Second
FMCG	Fast Moving Consumer Goods
FN	False Negatives
FP	False Positives
FPN	Feature Pyramid Network
FPS	Frame Per Second
FYP	Final Year Project
GAN	Generative Adversarial Network
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HSV	Hue Saturation Value
IBM	International Business Machine
IDE	Integrated Development Environment



IoU	Intersection over Union
ISCOS	Intelligent Self Checkout System
JSON	JavaScript Object Notation
LCD	Liquid Crystal Display
LDR	Light Dependent Resistor
LED	Light Emitting Diode
MAC	Memory Access Cost
mAP	Mean Average Precision
MCO	Movement Control Order
MIPI	Mobile Industry Processor Interface
POS	Point of Sales
PR	Precision-Recall
QR	Quick Response
R-CNN	Region Convolutional Neural Network
ReLU	Rectified Linear Unit
ResNet	Residual Network
RFID	Radio Frequency Identification
RGB	Red Green Blue
ROI	Region of Interest
ROIAlign	Region of Interest Aligning
ROIpool	Region of Interest Pooling
RPC	Retail Product Checkout
RPN	Region Proposal Network
SE	Squeeze and Excitation
SIFT	Scale Invariant Feature Transform
Softmax	Normalized Exponential Function
SPP	Spatial Pyramid Pooling
SSD	Single Shot Multibox Detector
SURF	Speed Up Robust Features
SVM	Support Vector Machine
TP	True Positives
USB	Universal Serial Bus
UTF	Unicode Time Format
VGG-16	Visual Geometry Group-16

VIA	VGG Image Annotator
VOC	Visual Object Classes
VRAM	Video Random Access Memory
WA	Wait Area
XML	Extensible Markup Language
YOLO	You Look Only Once

**LIST OF APPENDICES**

Appendix A: Image Synthesis Script	94
Appendix B: Image Augmentation Script	99
Appendix C: YOLOv3 Training and Evaluation Codes	102
Appendix D: RetinaNet Training and Evaluation Codes	103
Appendix E: GAN Training Script	104
Appendix F: YOLOv5 Training Script	105
Appendix G: Light Weight Models Architectures	106
Appendix H: Tables	109

## CHAPTER 1

### INTRODUCTION

#### 1.1 General Introduction

In general, retail can be defined as commercial activities that involve direct selling of merchandise to consumers at a specific point of purchase (Farfan, 2020). Among all the range of merchandise available in the market today, most of them are generally categorized under Fast-Moving Consumer Goods (FMCG) which possess several characteristics such as high consumer demand, common availability and associated with wide variations. With that, the industry requires high capacity of manpower in supply and distribution chain which encompass checkout system and stock management for the purpose to accommodate high requirement of FMCG products.

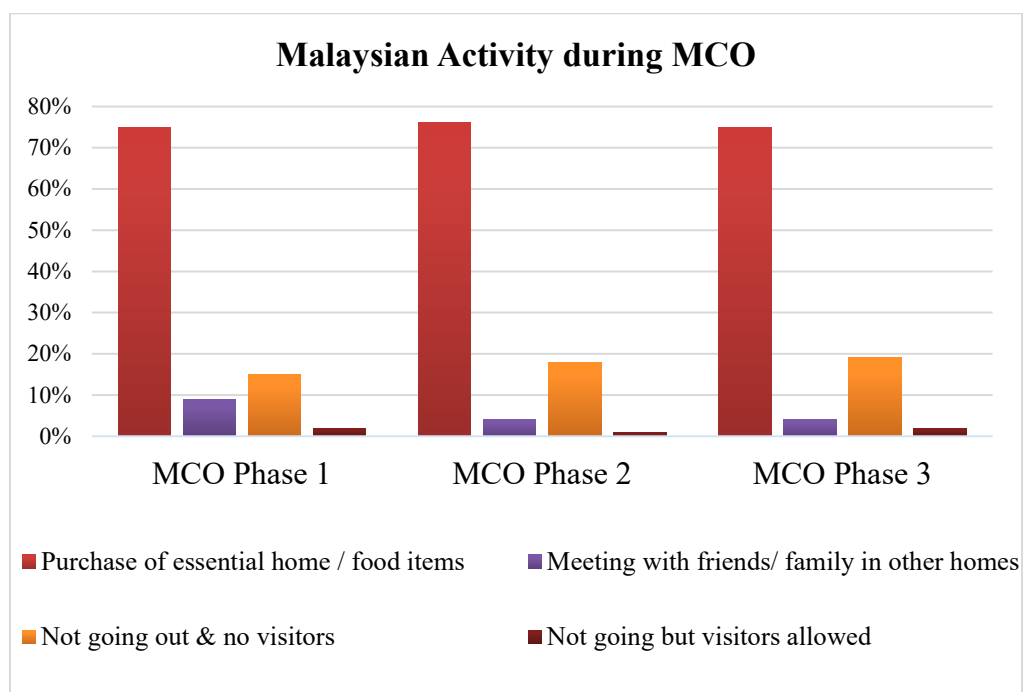


Figure 1.1: Malaysian MCO Activity 2020 (Hirschmann, 2021)

Ever since the first confirmed COVID-19 case reported on 25<sup>th</sup> January 2020, infection chain of the virus has begun and led to government's imposition of Movement Control Order (MCO) on 18<sup>th</sup> March 2020. As a result, Malaysian were forcefully grounded from non-essential activities. Nevertheless, they are

permissible to pursue for own living necessities generally classified under FMCG. This can be validated through the statistics provided by Hirschmann (2021) in Figure 1.1 which indicates that main purpose of going out was to make purchase of essential needs. Thus, with the increased flow of FMCG in the retail industry, chances of human involvement at point of purchase will be elevated in return. This could possibly lead to high risk of COVID-19 infection due to close interaction between consumers and employees.

Fortunately, according to O'Mahony et al. (2020), in present days where rapid advancement computing power and other device capabilities, computer vision based applications have been enhanced significantly in terms of performance along with cost-effectiveness. This has open up possibility of implementing computer vision-based checkout system in retail industry especially when the FMCG are involved. Specifically, by implementing computer vision-based solution to replace the traditional checkout operation, existing vulnerability such as high manpower requirement can be overcome as the system is capable to handle large volume of goods flow and will contribute to reduce human labour in the conventional checkout process such as barcode scanning or manual data entry. Additionally, the computer vision-based checkout system will also play a vital role in minimizing human to human interaction during COVID-19 pandemic. Therefore, a computer vision-based solution will serve as an ideal alternative to revolutionize the existing form retail checkout operation.

## **1.2 Importance of the Study**

The result in this study might impose significant impact in the field of retail checkout by introducing a software prototype of computer vision-based checkout system that is capable to perform product recognition and computation of product price in a real-world checkout counter scenario that can potentially reduce the requirement of human employee in checkout process along with the risk of developing COVID-19 virus infection chain during the pandemic.

Additionally, this project explores the possibility in simplifying data preparation process for deep learning-based product recognition model by designing a framework that can generate vast amount of training data without

requiring human involvement in preparing and capturing different images for model training session.

Furthermore, this project delved into the deployment of product recognition model that can run efficiently on edge devices, reducing the requirement of large computing resources in performing product recognition.

### **1.3 Problem Statement**

As COVID-19 pandemic strikes the world, retail industry has been severely impacted especially in their daily operation. During the pandemic, multiple policies were introduced by government which resulted in staff shortage in store since close contact within 1 meter will inflict higher risk of being infected by COVID-19 virus as stated by Ministry of Health Malaysia (2020). The situation causes the remaining employees to be burdened with higher workload especially in some essential tasks such as checkout process that generally involves product scanning, packaging and payment handling. When store personnel are insufficient to handle the operation, situation may go worse by resulting in clustering of customers or long queue in the store and eventually contribute to chain infection. Therefore, a smart cashierless checkout system is required to reduce the requirement of human involvement in checkout process.

Despite there are smart retail solutions such as Radio Frequency Identification and on-shelf computer vision checkout, but they require overhaul of store layout and are normally associated with high cost, which is unfeasible for small-scale retail stores with less business capital. Thus, a computer vision-based checkout system that can be easily implemented in existing retail stores is highly required.

Meanwhile, in technical perspective, computer vision application that utilizes deep learning model will have high demand of annotated training images that closely resembles the actual scenario since its performance is highly dependent on the amount and quality of training data. However, manual acquisition of training data can be challenging as it consumes time and easily affected by human error, especially when involving retail products that will change rapidly in terms of appearance and sub-categories (Wei et al., 2020). Therefore, a framework that can generate reliable training data will be advantageous for computer vision-based checkout system.

#### **1.4 Aim and Objectives**

The ultimate goal of this project is to construct a software prototype for computer vision-based cashierless checkout system on edge devices using state-of-the-art deep learning models at its core so that small retail can easily implement it at their checkout counters to minimize the concern of staff shortage and avoid the spread chain of COVID-19 virus during traditional face-to-face checkout process. The system is expected to accomplish several sub-objectives as below:

- (i) To construct an image pre-processing framework that is able to simulate actual checkout counter scenarios with minimal human involvement.
- (ii) To construct a deep learning-based product recognition algorithm using state-of-the-art object detection models.
- (iii) To optimize product recognition models for inference on edge devices
- (iv) To construct a software prototype that can compute total price based on recognized products.

#### **1.5 Scope and Limitation of the Study**

As this project involves the construction of software prototype for computer-vision based checkout system on edge devices, it will cover the collection and review of retail product dataset, followed by preliminary benchmarking of several deep learning models used in literatures, improvement of best performing model through light-weight backbones, TensorRT optimization and deployment on edge device.

However, this project is associated with several limitations. One of the limitations represents the hardware constraint. As this project is based on deep learning that involves intensive multiplication of matrices in parallel, powerful Graphics Processing Unit (GPU) will be required. Despite an online Integrated Development Environment (IDE) known as Google Colaboratory provides a NVIDIA Tesla T4 GPU with 16GB Video Random Access Memory (VRAM), but model training time is constrained by the platform's maximum GPU acceleration of 12 hours. Thus, to prevent force termination of training session, hyperparameters

of each model need to be reduced accordingly. This problem can be prevented through monthly subscription of Google Colaboratory Pro+ at \$49.99.

Besides, this project is also restricted by time constraint, some components of the cashierless checkout system framework can be further optimized to ensure its practicality and robustness in actual implementation. Each process can be automated to further reduce the requirement of human involvement in training the product recognition model.

## **1.6 Contribution of the Study**

The software prototype proposed in this study may serve as a conceptual solution to small-scale retail stores in transforming their daily operation into a semi-smart retail where less human employee will be required during the in-store checkout session, thus reducing the risk of COVID-19 virus infection during the process.

Besides, the software prototype is convenient and cost-effective since it can be deployed on edge devices with low power consumption, along with user-friendly features such as automatic update of product recognition model and product prices through database, which can reduce the overall cost for implementation in existing small-scale retail.

Additionally, the proposed deep learning model training framework may help to simplify training data preparation and annotation process since it is able to generate large number of high-quality training data from limited raw images of retail products, thus reducing the requirement of domain experts in the store.

## **1.7 Outline of the Report**

This report will be mainly comprised of 5 chapters. The first chapter covers the general introduction of retail stores and computer vision-based checkout system, followed by the aims and objectives of this project. At the same time, scope and contribution of this project will be described.

In Chapter 2, literature review of existing works in the retail product recognition will be included along with the analysis of their suitability to be implemented in this project.



As for Chapter 3, overall framework of the cashierless checkout system will be described. After that, methodology used in developing the software prototype will be explicitly explained in chronological order.

Chapter 4 summarizes all the results that were collected and tabulated throughout this project along with justification based on the theoretical knowledge from literatures.

Finally, in Chapter 5, this project will be concluded based on the objectives that were achieved, followed by recommendations of work that can be implemented in the future.

## CHAPTER 2

### LITERATURE REVIEW

#### 2.1 Existing Cashierless Checkout Approaches

According to Zhong (2021), advancement of computation technologies has facilitated rapid growth of retail industry, allowing the retail business to shift their traditional operations that requires significant work forces towards an advanced form by adopting state-of-the-art tools and techniques. Cashierless stores represent a concept that elevates the in-store shopping experience while reducing employment cost for the store by automating checkout process. In the field, there are two main approaches being adopted in cashierless retail which are Radio Frequency Identification (RFID) and computer vision respectively.

##### 2.1.1 Radio Frequency Identification (RFID)

By referring to Amsler and Shea (2021), RFID can be generally defined as a wireless communication based object identification method that utilizes electromagnetic or electrostatic coupling at specific radio frequencies within the electromagnetic spectrum.

In a fundamental RFID system, four main components are required to initiate data communication which are transponders, transceivers, antennas and reader interface layers. As stated by Sweeney (2010), transponder is simply known as a tag that will incorporate unique information in integrated circuit or chip to allow identification of product. It also comes with various form factors such as card, tag or labels to accommodate different applications. Furthermore, the tag can be categorized as active tag that requires power source and passive tags that can power itself by drawing energy from electromagnetic field emitted by RFID reader and provide response with its information. On the other hand, a transceiver is generally known as RFID reader which is aimed to handle radio communication with the tag through antennas and relay the tag information for processing. Lastly, a reader interface layer or middleware is required to identify the item through tag signals and interact with software systems such as inventory, logistics or point of sales (POS).

This technology was applied by Bocanegra et al. (2020) in their RFGo system, a RFID-based self-checkout system for apparel stores. The hardware setup included a divider placed apart from each other to form a checkout area (CA) and wait area (WA) where customer will queue up and walk between the divider with their items during checkout session. To ensure wide coverage of tag-reading area, Multiple antennas were deployed at inner surface of divider and beneath the floor. All of them were connected to a custom-built RFID reader that is capable in processing signal from antennas simultaneously. The collected tag information will be passed to a neural network-based classifier to determine whether the products are within the checkout area before proceeding to price computation.

Besides, similar approach was experimented by Panasonic (2018) in Fukuoka, Japan where the solution also comprised of a walk-through checkout lane with RFID readers and barcode scanners. The simplified shopping experience allows customers to enter shop after scanning their prepaid card and leave the store by walking through the checkout lane with products labelled with passive RFID tags. Products in the shopping bag will be identified through RFID reader in the divider of checkout lane and payment will be done directly. As a result, checkout efficiency is increased while reducing human involvement in the shop especially during the pandemic where social distancing can be important.

### **2.1.2 Artificial Intelligence (AI) and Computer Vision**

Another major approach of cashierless retail checkout is based on Artificial Intelligence (AI) and computer vision. As claimed by Gollapudi (2019), AI can be defined as a field of computer science in enhancing capability of computer to perform tasks that require human intelligence. Moreover, as a subfield of AI, computer vision simulates human vision on computing devices by enabling retrieval and interpretation of useful information from visual inputs like images or video feeds. This allows computer to perform different tasks that require visual perception such as image classification, object detection and object tracking. (International Business Machine Corporation (IBM), n.d.).

As claimed by Wei et al. (2020), a computer vision-based product recognition system generally involves 5 steps where the system is commenced

by obtaining an image through cameras. Afterwards, the raw image will be pre-processed through techniques like image segmentation and enhancement to remove unnecessary information in the image so that the feature extraction can be performed effectively in the following step. After the process, unique characteristics of image that are known as features will be mapped as feature vector with a lower dimension before they are being classified accurately based on pre-trained decision rule. Finally, predicted product category will be provided as an output for further manipulation according to different applications like price computation or stock management.

There are multiple cashierless checkout systems developed based on AI and computer vision throughout the years and Amazon Go represents one of the approaches that has been launched commercially in year 2018. Amazon (2016) stated that its unmanned store involved combination of AI, computer vision and sensor fusion technologies to facilitate smart retail experience. In the store, multiple closed-circuit television (CCTV) cameras were installed to provide visual inputs to neural networks for recognition of customers' action and product detection. At the same time, sensors fusion that comprised of weight sensors were implemented to detect the products added or picked from shelves while infrared sensors were positioned to detect the presence of hands. As a result, recognition performance and reliability can be improved while simplifying shopping experience. User can enter the store by scanning Quick Response (QR) code at entrance, grab in-store products and leave the store to complete their purchase.

## **2.2 Datasets of Retail Products**

Before proceeding to selection of deep learning model for retail product detection, dataset is mandatory and serves as a common ground for benchmarking the performance of algorithms while encouraging the development of high-quality deep learning solutions in the field. However, training of deep learning-based object detection models will generally require large amount of annotated data which can be labour-intensive and time consuming. Furthermore, uniqueness of retail checkout scenario such as intraclass variation and product occlusion need to be taken into consideration to train a robust product detection algorithm (Wei et al., 2020). Fortunately,

several datasets are made available publicly for benchmarking of deep learning-based retail checkout algorithm and they are widely discussed in the field. These datasets are generally prepared for two checkout situations which are on-the-shelf and checkout counter. Therefore, the following subchapters will involve description of datasets based on both scenarios and their techniques in tackling peculiarity of retail checkout scenario.

### 2.2.1 Retail Product Checkout Dataset (RPC)

Retail Product Checkout (RPC) dataset was published by Wei et al. (2019) as a large-scale dataset with 83739 images in total. 53739 of them are training images that only contain single-product images and 30000 validation images where each of them contains multiple products placed on checkout counter. All training images are captured at a high resolution of  $2592 \times 1944$  while validation images are captured with resolution of  $1800 \times 1800$  under controlled image acquisition environment. The dataset comprises 200 object classes of products from grocery stores in China as in Figure 2.1 and they are further divided into 17 super-categories such as instant noodles, drink, milk, canned food and candy. Each of the images is annotated with chinese product names, meta-categories and location of bounding boxes that is stored in Javascript Object Notation (JSON) files and the data structure complies with Common Objects in Context (COCO) standard by Microsoft.



Figure 2.1: RPC Dataset: Training (a) and Validation Images (b)

The dataset is associated with several characteristics that will improve the evaluation of product recognition models. Firstly, images are captured in plain background, hence no image processing is required before instance segmentation or directly fed to object detection model for training. Moreover, the dataset is prepared to simulate actual checkout situation at a checkout counter where products are placed in random orientation and involve multiple number of product instances. This helps to evaluate the proposed model for its robustness for actual implementation. Besides, validation images have been split into three clutter levels based on number of product instances which helps to further evaluate capability of model in handling large number of detections at once.

However, the dataset has some shortcomings where all the training images are prepared in such a way that each product are rotated 360 degrees and an image is captured every 9 degrees at different perspectives. It leads to a scenario where some products are having common features such as top view of canned drinks and canned food. This will cause the model to make false detection and affect its performance evaluation. Furthermore, each product name in dataset provided in Chinese characters in Unicode format which may be cause difficulty in research of product detection model due to different encoding format.

### **2.2.2 Densely Segmented Supermarket (D2S) Dataset**

There is another retail product dataset released by Follmann et al. (2018) known as MVTec D2S. It contains 21000 images of local groceries and everyday products that belong to 60 categories from Germany. Each image is acquired at a resolution of  $1920 \times 1440$  under controlled setup and environment. 4380 of the images are for training purposes which only involve single category of products, placed at different orientation in three types of backgrounds and lighting conditions. On the other hand, there are 3600 validation images and 13020 test images that consist of single or multiple objects from different categories placed in close proximity and occlusion is applied. Samples of training and validation images are shown in Figure 2.2. At the same time, the dataset comes with complete annotation that contains product name, instance mask for instance segmentation and bounding boxes for object detection. All

annotations are stored in multiple JSON files and complied to COCO format which ease the extraction of labels.

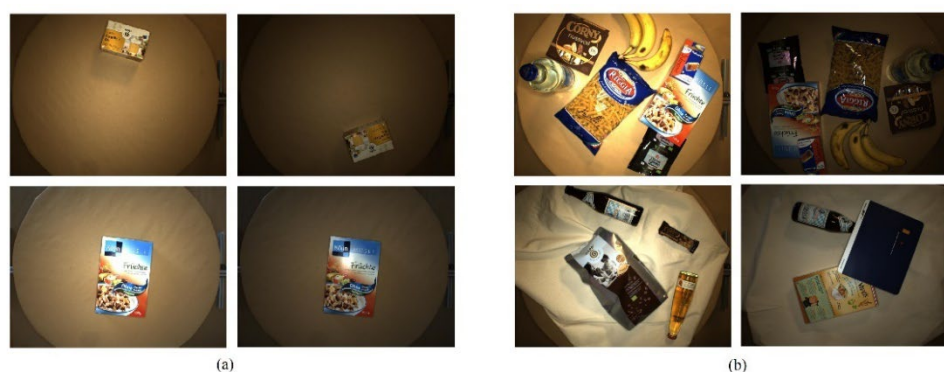


Figure 2.2: D2S Dataset: Training (a) and Validation Images (b)

The dataset comes with several characteristics which will help in assessing and comparing the performance of object detection models. First of all, most of the products are captured in plain background which helps mask extraction for instance segmentation and detection in certain models such as Mask Regional Convolutional Neural Network (R-CNN). Besides, variation of lighting and occlusion is considered in the dataset which helps to train and evaluate the model in terms of their robustness to changes of lighting. Moreover, the dataset introduces augmented data forming 10000 additional synthetic images that contain multiple categories of overlapped products in random orientation. With the presence of synthetic data that resembles the actual checkout situation, all models can be directly trained and assessed in depth without utilizing the data augmentation algorithm which can be time consuming. On the other hand, all the annotations are in English and comply with Unicode Time Format-8 (UTF-8) encoding format.

However, the dataset is associated with a disadvantage where the ground truth annotations for test images are not publicly released for evaluation. This will in turn causes difficulty in performing fair comparison with the baseline results in the paper. Despite the prediction results can be submitted in JSON files to author for evaluation, but it is not favourable due to time constraint of the research.

### 2.2.3 CAPG Grocery Product Dataset (CAPG-GP)

This checkout related dataset has been published by Geng et al. (2018) aimed to tackle intraclass variation of products. The dataset includes 102 classes of shampoo products in China, and they are further segregated into 177 sub-classes according to intraclass variants such as product volume and package design. This forms 351 images in total where 177 of them represents training images from and 234 images are used as test set. As shown in Figure 2.3, training images with plain white background are taken from e-commerce sites and having different image resolution, ranging from 200 to 600 pixels. On the other hand, test images are collected directly from product shelves of two different stores using smart-phone cameras at a resolution of  $4032 \times 3024$ . The approach helps to create lighting and background variation which is suitable for benchmarking. As for the annotations, information such as classes and bounding boxes for each image are provided in text files, hence no manual labelling is required. However, product names are not included in the dataset which will cause difficulty in benchmarking as the dataset involves many product classes.



Figure 2.3: CAPG-GP Dataset: Training (a) and Test Images (b)

### 2.2.4 Freiburg Groceries

In the paper by Jund et al. (2016), another retail product based dataset is introduced, which is known as Freiburg Groceries. The dataset comprises of 5021 images, involving 25 classes of local groceries from Germany. The datasets are divided into 4947 train images and 74 test images. Each training image contains single or multiple instances of one category captured through



smartphone cameras at multiple environments at varying aspect ratio. Thus, zero-padding is performed by adding pixels with value zero around the border of images as an approach to resize them to 1:1 aspect ratio before they further resized to  $256 \times 256$  pixels. According to Hashemi (2019), zero-padding may help in improving computational efficiency of a neural network as the calculations are speeded up through deactivated convolutional unit by zero pixels. On the other hand, each test image is captured at resolution of  $1920 \times 1080$  and involves multiple instances from different categories on their corresponding shelves. Some samples of train and test images are illustrated in Figure 2.4.



Figure 2.4: Freiburg Dataset: Training (a) and Test Images (b)

However, the dataset comprises of several shortcomings. First, all training images are pre-downscaled to low resolution ( $256 \times 256$  pixels) which may be inappropriate to be applied to some state-of-art object detection models that require higher input image resolution. This is because training of deep learning models is depending on combination of object features such as color and edges. Therefore, decrement of input image resolution will imply negative impact to the model performance due to pixelation of object outline and blurring of edges despite color features are still intact (Seals, 2019). On the other hand, bounding box information is not included in the annotation files which may cause inconsistency during comparison between different product detection approaches. Besides, all images are annotated based on generic product type such as cereal, chocolate and milk, which is incompatible with actual checkout scenario that requires detection of exact variation of products.

### 2.2.5 Dataset Comparison

Based on dataset summary in Table 2.1, RPC and MVTec D2S Dataset are more relevant in this project compared to CAPG and Freiburg Groceries because they are prepared for checkout counter scenario, which suits the purpose of this project. However, MVTec D2S Dataset is considered incomplete since the test set is not published in their repository, imposing difficulty for evaluation of product recognition models. In contrast, RPC dataset is complete with COCO format annotation, but lighting variation is not considered, and its size is relatively large with 83739 images in total, which will demand for hardware with higher computing power and can possibly increase the training time required for product recognition.

Table 2.1: Characteristic Table of Publicly Available Dataset

Characteristics		MVTec D2S	RPC	Freiburg Groceries	CAPG - GP
Categories		60	200	25	177
No. of instances	Train	Vary	Single	Multiple	Multiple
	Val / Test	Multiple	Multiple	Multiple	Multiple
Resolution	Train	1920	1592	256	Vary
		×	×	×	
	Val / Test	1440	1440	256	
		×	×	×	×
Quantity	Train	14380	53739	7425	177
	Val / Test	13020	30000	995	234
Scenario		Counter	Counter	Shelf	Shelf
Background		Distinct	Same	Distinct	Distinct
Lighting variation		✓	✗	✗	✓
Occlusion of products		✓	✓	✓	✓
Completeness		✗	✓	✗	✗
Intraclass variation		✓	✓	✗	✓

Therefore, MVTEC D2S dataset will be pre-processed before using it for preliminary benchmarking and comparison with other existing models which adopted the dataset. Additionally, another dataset will be prepared for this project based on the characteristics of both D2S and RPC datasets by taking lighting difference, product occlusion and intraclass variation into consideration.

### **2.3 Image Synthesis**

In general, size of training data can be a constraining factor in achieving a well-performed object detection algorithm. As reviewed by Wei et al. (2020), deep learning based retail product recognition approach suffers from data scarcity by having fewer images per class compared to common object datasets such as COCO and PASCAL Visual Object Classes (VOC). Hence, data augmentation needs to be adopted to generate vast number of artificial images to boost the performance of model across wide range of object classes. According to Saxena (2020), data augmentation represents a strategy in increasing data diversity by introducing variation to existing data without manual collection of new data. Several data augmentation techniques have been implemented in the domain of retail product detection, which includes basic image manipulation, mask-based synthesis and Generative Adversarial Network (GAN)-based data augmentation.

#### **2.3.1 Conventional Data Augmentation**

Conventional data augmentation approach encompasses fundamental manipulation of images such as geometric transformation, flipping and color space transformation, cropping, rotation, translation and noise injection. This approach has been utilized by Rigner (2019) in his retail product detection solutions.

In the paper, multiple traditional data augmentations like scaling, noise injection, brightness adjustment, and mirroring were applied to his custom dataset before it was fed to Mask R-CNN for training. In the research, scaling of 50 to 150% was utilized on images and annotations. Besides, Gaussian noise was used as a form of noise. Furthermore, brightness was manipulated by multiplying each image pixel by a factor of 0.5 to 1.5. Despite shearing was considered in the initial research but it was removed due to its negative impact on detection performance. With the extended dataset, Mean Average Precision

(mAP) performance of the proposed Mask R-CNN model was improved to 60.1%, yielding a performance leap of 20.8% compared to the model trained only by raw images with a mAP of 39.3 %.

### **2.3.2 Mask based Synthesis**

In the actual practices of retail product recognition on shelves or checkout counters, occlusion and overlapping of products can be a common yet challenging phenomenon for deep learning-based models. Hence, in order to simulate the scenario, mask-based approach has been widely adopted by several researchers as a form of data augmentation. As the name suggests, mask-based image synthesis involves the extraction of products with corresponding masks before they are used to generate synthetic images with multiple instances per image.

The first mask-based augmentation approach was proposed by Yi et al., (2019) in developing their occlusion simulation algorithm for retail product detection framework. Prior to the augmentation, their dataset which consisted of single and multiple product images was collected under a controlled environment with a plain, dark background. Subsequently, the images were fed to a bounding box extraction algorithm where multiple bounding box proposals will be generated through selective search before small, distorted proposals below a threshold value were filtered to obtain accurate bounding boxes. All objects were then cropped according to boxes and stored as patches in the database. By feeding a random patch and a raw image as inputs, augmentation can be performed through an occlusion simulation algorithm. Thresholding was applied on the patch to obtain a binary mask for background removal. The patch was then cropped and placed on the original image which resulted in a synthesized image with a realistic product occlusion. In brief, the framework can be represented in Figure 2.5. The approach was proven to be effective on Faster R-CNN-based retail product recognition algorithm by achieving a high mAP of 84 % on actual products placed in closed proximity on a conveyor.

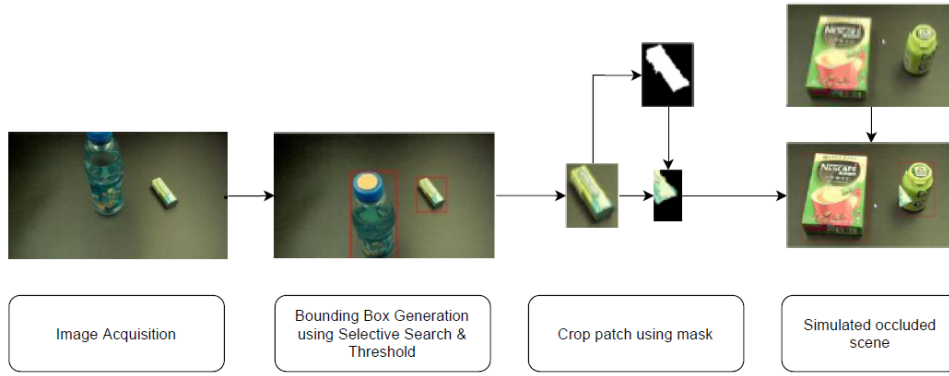


Figure 2.5: Mask-Based Image Synthesis

On the other hand, Koturwar, Shiraishi and Iwamoto (2019) also introduced a similar approach in expanding the dataset for their automated checkout system. Before the mask extraction process, images of individual products were captured above a Liquid Crystal Display (LCD) platform that will display red, green, and blue (RGB) plain backgrounds. It was claimed that the method will aid the robust extraction of product masks especially when it was having similar color with background compared to the traditional method which utilized fix-colored background. Sequentially, a product mask was generated by calculating pixel-wise standard deviation across all background colors, which is represented through formula 2.1 where  $N$  represents the number of images of different backgrounds,  $c_{i,p}$  is the image intensity at each pixel  $p$  in image  $i$ . On the other hand,  $\mu_p$  and  $\sigma_p$  are the average and standard deviation of each pixel  $p$  for all images of the individual product.

$$\sigma_p = \left[ \frac{\sum_i (c_{i,p} - \mu_p)^2}{N} \right]^{\frac{1}{2}} \quad (2.1)$$

With the computed standard deviation for all pixels, an accurate mask was then generated by thresholding the standard deviation based on the concept where products will have lower standard deviation because their pixel values will not be affected by changes of background color. With the product mask, synthetic training images with product occlusion can be generated by first randomly selecting a base image. Subsequently, multiple products were placed randomly according to overlap index that was a threshold calculated through

Intersection over Union (IoU) between product masks. The overall framework will be as shown in Figure 2.6.



Figure 2.6: Mask extraction and Merging for synthetic data generation

(Koturwar, Shiraishi and Iwamoto, 2019)

The approach was proved to be effective to imitate the actual placement of products by a human during the checkout process. By feeding the synthetic dataset into Faster R-CNN with Residual Network (ResNet) backbone with 101 layers known as ResNet-101, their proposed method achieved a high precision-recall of (0.84, 0.98). Compared to the unaugmented dataset with the precision-recall of (0.60, 0.67), the performance was extensively improved by 0.2. At the same time, the result was compared with a synthesized dataset through the cut and paste approach and there was a slight improvement in precision-recall of 0.04 and 0.06 respectively.

### 2.3.3 Generative Adversarial Network (GAN)

Despite conventional and mask-based data augmentation methods can significantly boost detection performance of occluded retail products with limited training data, but they are incapable of simulating a natural checkout condition such as shadows patterns and lighting conditions. Therefore, according to Wei et al. (2019), GAN-based models such as CycleGAN allow a realistic rendering of synthetic images known as image-to-image translation, which increase correlation between synthesized images and training images to enhance detection accuracy.

Such approach has been demonstrated in the papers by Li et al. (2019) and Wei et al. (2019) in creating realistic images similar to the actual checkout counter condition. RPC Dataset as mentioned in Section 2.2.1 which only contains training images of single products was adopted. Synthesis of images was initiated with the creation of checkout images with overlapping products

through a mask-based approach on a plain background image. At the same time, the area of each product mask was ensured to exceed a pre-defined threshold value to simulate the realistic placement of bag-like products on a checkout platform. Subsequently, the synthesized images were fed into a pre-trained CycleGAN so they can be rendered to look like an actual checkout scenario with the presence of shadows and realistic lighting conditions. The method was proved to be practical as the detector trained with original and translated images was able to attain a high mAP of 96.57%.

## **2.4 Detection Algorithm**

When images of retail products have been acquired and pre-processed through conventional data augmentation or GAN-based algorithms, they will be applied to a detection algorithm for product recognition so that prices can be computed according to the detected products.

In the past decades, computer vision has already been widely adopted in the research field of retail product recognition and checkout system. According to Wei et al. (2020), computer vision-based applications in the field was started with conventional methods in the early days before the applications were expanded to deep learning-based approach. However, conventional computer vision approaches are still being actively engaged in the research field for deployment on devices with constrained computing capabilities such as embedded systems.

### **2.4.1 Classic Computer Vision Techniques**

As stated by IBM Cloud Education (2020), machine learning represents a subfield of AI which mainly studies the methods in replicating human thinking capability on computing devices with the use of data and algorithms. In earlier days when deep learning was largely constrained by computing technologies, most of the computer vision applications including retail product recognition were achieved through traditional machine learning techniques. As stated by O'Mahony et al. (2020), a traditional approach to object detection mainly involves the combination of a conventional computer vision technique known as feature extraction and a traditional machine learning classifier to form a complete framework. Feature extraction can be defined as a process of encoding

meaningful information of an image into a vector known as features by utilizing a feature descriptor algorithm. Once the features are extracted, object detection can be generally done through feature matching or machine learning-based classifier. Feature matching involves searching for similar features in another image. If a high number of features exist in another image, the image is said to contain specific objects. In contrast, a machine learning-based classifier can accurately predict object classes after being trained with the corresponding dataset.

In the field of retail product recognition, it was found that Scale Invariant Feature Transform (SIFT) and Speed Up Robust Features (SURF) algorithm were commonly used for feature extraction and machine learning algorithms such as Adaptive Boosting (AdaBoost) and Support Vector Machine (SVM) were used for product classification and recognition. The following subchapters will summarize traditional approaches by other researchers.

#### **2.4.1.1 SURF + Feature Matching**

In the retail product recognition research conducted by Moorthy et al. (2015), SURF algorithm was utilized to realize product detection and positioning in retail store shelves.

The system can be decomposed into 5 steps, starting with the input of two images from a user where one of them acted as a reference image of a specific product while another image served as a target image which was captured through a camera device. Then, both images were converted to grayscale images to improve computational efficiency. At the same time, the reference image was further cropped to remove irrelevant background to allow the detection of multiple products within the target image. The following step was the feature extraction through SURF algorithm due to its invariance to lighting, product scale and contrast. As a result, each image will produce a set of features that were comprised of two elements known as characteristics points and descriptors. Then, the comparison was done between the extracted features and all matched points were recorded so that the transformation relationship between the points can be computed to obtain the products' exact location in the target image. The whole process was iterated multiple times until all products



were found based on SURF features. For visualization, all found products were marked in bounding boxes and the overall product count was displayed.

Performance of SURF features was demonstrated by author through different images of actual shelf situation. Despite there were no quantitative results being tabulated in the paper, the author has demonstrated that the conventional algorithm was able to tackle intraclass variation in terms of packaging color and achieved accurate localization of empty space on shelf despite with an average execution time of 190 seconds. However, such approach was proven to be highly dependent on completeness of image as it will decrease number of generated SURF features which will result in rejection during matching process.

#### **2.4.1.2 Haar Features + Adaboost & SIFT + SVM**

An approach that utilized conventional feature descriptors can be seen in the research by Varol and Salih (2015) regarding the recognition of tobacco products placed on shelf. They have constructed a two-stage framework that involved product segmentation and brand recognition.

The algorithm began by first taking an input image with tobacco products on shelf-to-shelf boundaries detection. In this process, a histogram was generated to project the products in y-axis direction which allows the identification of product distribution on the shelf. After applying Gaussian filter for noise removal, position of products can be differentiated as products will have peak value while shelf as non-product will generally have a low value. Besides, number of shelf space can be determined through the histogram to further determine the range of product height for segmentation of tobacco products.

In the segmentation process, a cascaded object detection module based on Viola-Jones object detection framework was adopted. The conventional module involved a unique image representation known as Integral Image and rectangle-like features known as Haar Features. At the beginning of the module, each pixel in the Integral image was computed by obtaining the sum of pixel values in x and y direction. Subsequently, sliding window was applied on the integral image and each region was used for Haar Feature extraction by computing the difference between sum of pixels in white rectangle and the sum

of pixels in black rectangle within the specific region. Each feature was then fed to a cascaded AdaBoost Classifier trained with Grozi-120 dataset. Through multiple layers of classifier with several types of additional thresholds such as minimum and maximum product height, irrelevant features were rejected by having values lower than pre-set threshold while best features that represent the products can be obtained and labelled with bounding boxes.

The following step was the brand recognition algorithm that involved logos of the detected regions. SIFT feature descriptor and Hue Saturation Value (HSV) color space were involved in this stage to represent shape and color information respectively. Each feature descriptors were then used to form a frequency histogram for computation of joint feature vector. Finally, classification can be performed through multi-class SVM.

The product detection module had achieved a relatively high recall of 0.94 and 0.75 by including the product height thresholds. Concurrently, the brand classifier has achieved classification accuracy of 92.3% by involving both SIFT and HSV feature descriptors.

#### **2.4.2 Deep Learning based Computer Vision Techniques**

According to Chauhan and Singh (2018), deep learning serves as a subfield of machine learning which is generally a study that explores and construct algorithms in allowing computing devices to learn from a given training data and perform prediction on unseen data. However, deep learning is more advanced as it tends to focus on deployment of Artificial Neural Networks (ANNs) which involve cascading of multiple layers of interconnected nodes known as neurons that are aimed to imitate human brain in processing the information and perform prediction as a solution to complex problems.

Specifically, for computer vision applications like object detection, introduction of deep learning along with the aid of computing power and memory capacity advancement has caused a remarkable change of research direction. As stated by Zou et al., (2019), starting from year 2014, object detection has been shifted from conventional machine learning approaches like Viola Jones detectors towards deep learning based detection methods as the techniques will help to overcome the performance bottleneck of traditional computer vision approach while requiring less domain specific knowledge as

they are trained instead of being programmed. Moreover, the approach offers more flexibility as they are able to be re-trained to adapt to different dataset. Hence, in the field of retail product recognition, deep learning-based methods are widely adopted by researchers and plenty of work were published.

#### **2.4.2.1 Convolutional Neural Network (CNN)**

As stated by Khan et al. (2018), Convolutional Neural Network (CNN) represents the most popular deep learning architecture in the field of computer vision and its development plays an important role by contributing performance leap in visual recognition tasks like image classification, detection and localization. Hence, it is served as a backbone architecture for most of the deep learning-based object detectors in present days.

In general, CNN can be expressed as a deep learning algorithm that can study the spatial information in high-dimensional input data such as images or videos and dynamically assign weights and bias through backpropagation to effectively differentiate instances in the image. As stated by Yamashita et al. (2018), CNN is composed of 3 types of layers which are convolutional layers, pooling layers, as well as fully connected layers. A basic structure will typically involve multiple convolutional layers and pooling layers prior to fully connected layers. Convolutional layers are responsible to take in an input image and perform feature extraction through linear convolution operator and non-linear activation function whereas pooling layers are used for dimension reduction of feature maps generated by convolutional layers via max pooling or global average pooling to reduce trainable weights in the network. By taking in the feature vector, which is the flattened feature maps, fully connected layers will perform classification task based on the trainable weights.

Due to simplicity of the architecture, CNN can be easily deployed and customized to accommodate different applications including retail product detection. In the research conducted by Bukhari et al. (2021) which concerned a retail checkout system named as Automated Retail Checkout System (ARC), a self-modified CNN was applied to identify the product placed under a webcam. Prior to the training, data acquisition was performed to create a dataset with 100 classes of local items from Carrefour Pakistan. Each image was captured in a hood by involving single product placed at different orientation. All 31000

images were then spitted in a proportion of 65:25:10 to form train, validation and test sets respectively.

The custom CNN used in their research was a light-weight variant by only comprised of 7 layers in total where the first 4 layers were convolutional layers and max pooling layers stacked alternatively to each other. Each convolutional layer is associated with Batch Normalization and Parametric Rectified Linear Unit (PReLU) activation function. Then, the architecture was followed by 3 fully connected layers where PReLU were used for former two layers and Normalized Exponential Function (softmax) activation function was deployed at the output layer. PReLU was adopted for the CNN as it will help to overcome zero gradient issue for negative inputs in the conventional ReLU function. Hence, the network will be able to deal with high-complexity problems.

Prior to the training process, several hyperparameters were adjusted and they are referred to epochs of 100, batch size of 32 and learning rate of 0.001 that was set to be reduced by a decay rate of 0.96 and 0.75. This is because a decaying learning rate will encourage the network to learn complex patterns which can be useful in computer vision applications that typically involve real world datasets (You et al., 2019). Besides, a dropout of 0.1 was applied to prevent network from overfitting.

Through the training on Google Colaboratory, training and validation accuracy were 94.76% and 95.24%. The algorithm was further benchmarked with test dataset and achieved accuracy of 91.7%. However, classification performance was not ideal due to multiple misclassifications occurred for the products with glossy surfaces and packing of similar color.

#### **2.4.2.2 State-of-the-Art Models**

According to Davis (2021), state-of-the-art deep learning models can be referred as leading-edge neural network algorithms that obtain highest level of achievement at a specific point in time. In the field of object detection, as mentioned by Boesch (2021), multiple generic state-of-the-art architectures have been developed and improved throughout the years, especially from year 2014 to 2020, which gave rise to some popular CNN based object detection networks, such as R-CNN, You Look Only Once (YOLO) and Single Shot

Multibox Detector (SSD). Chronologically, these networks are continuously being improved which leads to numerous variants in the later years, as shown in Figure 2.7.

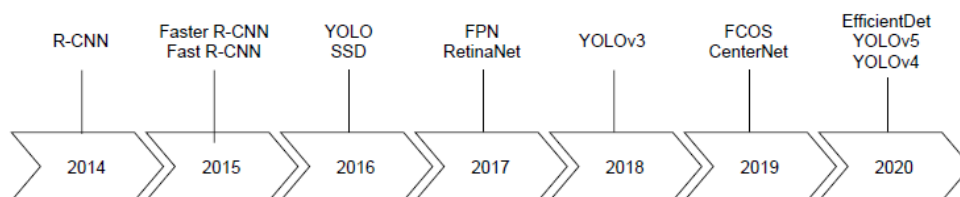


Figure 2.7: Development of State-of-the-Art model architectures (Boesch, 2021)

In general, according to statements by Zaidi et al. (2021), existing state-of-the-art object detection networks can be further divided into 2 types which are two-stage and single-stage architectures. Two-stage architectures such as R-CNN based algorithms perform object localization and classification separately by first generating Region of Interest (ROI) for each object before performing classification and regression of bounding boxes on each ROI. In contrast, single-stage architectures such as SSD, YOLO and RetinaNet can perform both tasks in parallel where it can directly generate bounding boxes and probabilities for each class in the input image with single forward-pass. Despite the architectures were having some drawbacks when first introduced, but active improvement has been conducted and multiple variants with performance improvement have been published. Hence, following subchapters will focus on recent architectures that are widely adopted in the field of deep learning which mainly cover Faster R-CNN and Mask R-CNN, YOLO family and RetinaNet.

#### 2.4.2.2.1 Faster R-CNN

As the third architecture of R-CNN models, Faster R-CNN developed by Ren et al. (2016) is referred as a region-based object detection model that performs object localization by generating bounding boxes around object of interest as well as classification of each object within the bounding box. As stated by Ananth (2020), compared to other models in R-CNN family, Faster R-CNN represents the most widely adopted State-of-the-Art model in deep learning

based object detection research. This is because Faster R-CNN proposed a solution known as Region Proposal Network (RPN) to overcome the computational constraint caused by Selective Search algorithm in its predecessor which is Fast R-CNN. By replacing the time-consuming Selective Search with RPN, region proposals can be generated faster which will in turn reduce the time required for model training and inference.

There are several research in the field that employed Faster R-CNN for their retail product recognition. In the research conducted by Liu et al. (2019), Faster R-CNN was applied to their computer vision-based checkout application in bakery store. By using their own dataset with 510 bread images that were split at a ratio of 452:55:3 through random selection, transfer learning of Faster R-CNN was done using a pre-trained Visual Geometry Group-16 (VGG-16) model. The training was iterated for 70000 steps and took 8.76 hours to complete. Based on the performance of 100 unseen images, their Faster R-CNN model achieved mAP of 100% compared to SSD that only achieved 9%. However, their SSD surpassed Faster R-CNN by 1.8s in terms of inference time.

Besides that, Faster R-CNN was also adopted in the paper by Koturwar, Shiraishi and Iwamoto (2019) which involved development of an automated POS System. By using image synthesis method mentioned in Chapter 2.3.2, 20000 artificial images were generated for training. Meanwhile, 3 different test sets were prepared to imitate 3 different stages of product occlusion in actual checkout, namely *Easy*, *Regular* and *Hard* with ratio of 300:600:10. The dataset was then used to train a Faster R-CNN with ResNet-101 backbone at a learning rate of 0.003,  $200 \times 10^3$  steps, confidence threshold of 0.8. After training, the model achieved precision-recall of (0.93, 0.99) and average of 0.86 when tested with real images under *Easy* scenario. As for *Normal* scenario, the model achieved precision-recall of (0.84, 0.98) and average IoU of 0.85. Concurrently, the model also performed well in *Hard* scenario with precision-recall of (0.61, 0.84) and average IoU of 0.78.

#### **2.4.2.2.2 Mask R-CNN**

Mask R-CNN is known as another variant of the R-CNN family that is widely used in object detection-related applications. The algorithm was developed by He et al. (2018) with the aim to perform object detection and generate their

corresponding segmentation mask simultaneously. It was achieved by introducing a custom Fully Convolutional Network (FCN) to further extend Faster R-CNN architecture with segmentation mask prediction in a pixel-to-pixel manner for the objects within the ROI. According to the author, FCN used in the algorithm is distinct to common FCN used for object detection in terms of activation and loss function. Softmax activation function and multinomial cross entropy loss function was replaced with sigmoid and binary loss function to improve segmentation accuracy and it was proved with increment of Average Precision (AP). Besides, another improvement was applied in Mask R-CNN to overcome the issue of misalignment of objects in the input image compared to quantized feature map generated by the original Region of Interest Pooling (ROI Pool). An additional layer known as Region of Interest Align (ROI Align) was proposed to replace ROI Pool to conserve the spatial information.

In the research field of retail product recognition, Rigner (2019) had utilized Mask R-CNN in their on-shelf self-checkout system. The project was initiated by preparing their retail product dataset. By utilizing the dataset with 486 images that were captured at a resolution of  $1920 \times 1080$ , the dataset was further increased to 2430 images through conventional data augmentation mentioned in Chapter 2.3.1. Each of them was labelled with class-wise segmentation mask and bounding boxes. The Mask R-CNN used in the research was based on ResNet-50 backbone for feature extraction purpose and it was benchmarked along with 2 single-stage object detector which were RetinaNet and YOLOv3 with same backbone. Transfer learning was applied by adopting the weights pre-trained with COCO dataset that was claimed to perform well in detecting low-level features. Thus, only network head which involved the convolutional layers for classification and segmentation needed to be trained. Nevertheless, hyperparameters used for training was not explicitly specified in the paper. Based on the evaluation of model at  $1920 \times 1080$ , Faster R-CNN achieved mAP of 72.3%, which was highest in contrast to RetinaNet and YOLOv3 with mAP scores of 71.8% and 51.9% respectively. As a trade-off, inference time of Mask R-CNN was the longest among all models due to high computational complexity.

On the other hand, Ning, Li and Ramesh (2019) also proposed their self-checkout system using Mask R-CNN. The dataset used for their

performance evaluation was MVTec D2S Dataset mentioned in Chapter 2.2.2. However, instead of applying the whole dataset, only validation set with 3600 images was endorsed and further split into training, validation and test set at a ratio of 80:10:10. As a result, a small dataset was formed with 2880 training images, 360 validation and test images. In the research, two Mask R-CNN algorithms used were based on two different backbone structures, namely ResNet-101 and ResNet-50. Similar to approaches in other papers, both models were pre-trained with COCO dataset for transfer learning purpose. Besides, several hyperparameters had been tuned for performance enhancement. Through random search, learning rate was set an optimal value of 0.002. Meanwhile, Adaptive Moment Estimation (Adam) optimizer was replaced with Stochastic Gradient Descent (SGD) because the former requires lower learning rate and result in longer training time. After the training for 10 epochs at an input resolution of  $512 \times 512$  due to GPU limitation, Mask R-CNN with ResNet-101 achieved mAP of 78.8% which was higher than ResNet-50 variant that achieved mAP of 75.3%. However, the author claimed that ResNet-50 was preferred for the application by providing a good training performance with slight degradation of accuracy. The model further achieved mAP of 84% by tuning 3 convolutional layers of network head.

#### 2.4.2.2.3 YOLO

According to Ohri (2021), YOLO is categorized under single-stage object detector that are generally used in object detection applications that emphasize speed, time and accuracy. As stated by Redmon et al. (2016) as its developers, the deep learning model represents a new approach to object detection by treating object detection as a regression problem, hence forming a unified model that allows bounding box prediction and computation of class probabilities in a single forward propagation of input image. The model will first divide the input image to  $S \times S$  grid. Subsequently, the grid cell which contains the center of object will need to predict its bounding boxes and class probabilities. Hence, the prediction will comprise of 5 components. 4 of them are normalized coordinates for center of bounding box as well as width and height of bounding box. The fifth element represents the confidence score that is calculated using



Intersection over Interference (IoU) to indicate accuracy of the predicted bounding box. These variables are being used with class probabilities throughout the model to perform prediction of object classes and bounding boxes localization simultaneously.

Several research had adopted the algorithm in YOLO family. In Wu et al. (2016) which introduced Intelligent Self-Checkout System (ISCOS), product recognition was done through a custom framework by cascading YOLOv1 and CaffeNet as shape detector and product classifier respectively. The system was designed in such a way that real time video feed will be supplied to YOLOv1, allowing localization of products and prediction of shape category. Then, product images were cropped according to bounding box and applied to CaffeNet in sequence for recognition of products. By using 317593 product data that was webcrawled from supermarket database and search engines at resolution of  $456 \times 417$ , the CaffeNet was trained with 317593 product data at for  $100 \times 10^3$  iterations, learning rate of 0.001, batch size of 128 and momentum of 0.9. Meanwhile, YOLOv1 as a shape detector was trained separately with 63271 images at resolution of  $448 \times 448$  or 750 repetitions. However, the dataset was annotated based on product shape instead of product category. Learning rate of the training was tuned to 0.0005, batch size of 64 and subdivisions of 8. In evaluation, the proposed method has achieved accuracy of 66.4% in single item scenario, 65.7% in two items scenario and 64.1% in multi-product scenario and their execution time was 69.6375ms for each inference of image.

On the other hand, in Oh and Chun (2020) which proposed a smart shopping cart, YOLOv3 was used to detect products present in the real-time video stream provided by Raspberry Pi mounted on cart. The dataset involved for the training was consisted of 5 types of bottled soft drink and 2800 images were captured and annotated through YOLO-mark software. YOLOv3 was then trained for 16200 epochs and several hyperparameters were configured for optimal performance. Learning rate was fixed at 0.001 while momentum and weight decay were set to 0.9 and 0.0005 respectively. In addition, confidence threshold was tuned to 0.6 to decrease false positive rate during training process. By testing the algorithm through real-time video stream for 10 times, mAP of 82.28% and AP per class was ranged from 66.7% and 88.9%.

#### 2.4.2.2.4 RetinaNet

Other than YOLO series, RetinaNet is found to be popular among researchers in the field of product recognition and checkout. As claimed by Lin et al. (2018) as its developers, RetinaNet was designed to resolve the performance bottleneck caused by extreme class imbalance between foreground and background during training of single-stage detectors such as SSD. It was achieved by introducing a new loss function known as Focal Loss which is an improved version of Cross Entropy Loss so that the network will assign more weights on hard examples such as targets that are partially visible while reducing the weight of background since it represents as easy example (Anwla, 2020). Thus, RetinaNet is able to achieve the optimal speed of one-stage detector while preserving the accuracy of two-stage detectors.

The implementation of RetinaNet in retail product recognition can be seen in the research conducted by Xie, Wang and Zhao (2021). RPC Dataset mentioned in Chapter 2.2.1 was chosen for training of all three models. As the training subset was incomplete with images to fully imitate the actual checkout scenario, GAN was implemented through the method stated in Chapter 2.3.3 to generate 30000 synthetic images with product occlusion and realistic lighting effect for training purpose. To allow comparison on the same ground, backbone structure of RetinaNet and Faster R-CNN was changed to ResNet-101 while DarkNet-53 was remained unchanged for YOLOv3 in the research. Despite training parameters was not mentioned in detailed, results were explicitly analyzed in the paper. At IoU of 0.75, RetinaNet had achieved the highest accuracy among three models with the mAP of 99.56% followed by Faster R-CNN with slightly lower mAP of 96.98%. In contrast, YOLOv3 achieved the lowest mAP with value of 82.32% in the benchmark. Besides that, RetinaNet was further tested with three different stages of product occlusion as prepared in the dataset. Based on a custom evaluation metric known as Checkout Accuracy (cAcc) that indicates the success rate of model in actual checkout process, RetinaNet achieved high cAcc of 91.65% at easy level but the value dropped to 82.3% and 71.65% for medium and hard level respectively.

### 2.4.2.3 Summary

According to Table K-1 which summarizes all related works in deep learning-based retail checkout system, it can be justified that most of the researchers were adopting three types of deep learning models, including R-CNN family, YOLO family and RetinaNet. Thus, RetinaNet will be used for further analysis along with two representative models under YOLO family which are YOLOv3 and YOLOv5.

The reason of R-CNN models like Faster R-CNN and Mask R-CNN are not chosen for further research is due to numerous research have been conducted and explicitly reviewed in the field of computer vision-based retail checkout system. Furthermore, due to the architecture as a two-stage object detector, R-CNN based models are complex, making it impractical for deployment on edge devices in this project. It can be justified in the bread recognition and checkout system conducted by Liu et al. (2019), their Faster R-CNN achieved a high mAP compared to single-stage detector like SSD but at a cost of long inference time of above 100ms. As for Mask R-CNN, Rigner (2019) stated that Mask R-CNN has similar drawbacks where its complex architecture leads to slow inference speed compared to other single stage detectors like YOLOv3 and RetinaNet. Moreover, according to the statement by Ning, Li and Ramesh (2019), performance of Mask R-CNN will be limited during the presence of distinct product size and occlusion. Thus, all two-stage object detectors will not be further evaluated in this project since the models cannot fulfill the requirements of a robust retail product recognition where overlapping of products, distinct product size commonly occur.

Thus, to ensure the product recognition model can be deployed on edge devices with limited computing resource, YOLO families and RetinaNet will be used for further analysis due to their high inference speed as single-stage detectors. This is justified by the research by Rigner (2019) and Xie, Wang and Zhao (2021) which showed that YOLOv3 and RetinaNet are advantageous in terms of inference time compared to R-CNN series with slight sacrifice of accuracy. Hence, for further testing of YOLO model series, YOLOv3 and YOLOv5 will be chosen as they are the representative models in current deep learning field. RetinaNet will also be added for comparison to justify its performance against YOLO models under actual checkout scenario with several

challenges such as overlapping products, intraclass variation and distinct product size.

## **2.5 Supportive Elements for Retail Checkout System**

In the current field of retail product recognition, most of the approaches emphasize on-shelf checkout rather than checkout conveyor. At the same time, as mentioned in Chapter 1.2, there are only a few research that involved hardware and software setup at the same time. One of them represents the ARC proposed by Bukhari et al. (2021), the checkout system setup used in their research was comprised of a checkout conveyor with a hood on top of it to prepare a controlled environment for image acquisition for training and inference of product images through a Logitech C310 webcam and Light Emitting Diode (LED) strips. The conveyor was actuated by a single-phase induction motor and controlled by an Arduino Mega 2560 through a 5V relay circuit. Additionally, a Light Dependent Resistor (LDR) was used to sense the presence of product in the hood and provide signal to computer for product recognition through pySerial library. Besides, a printer was connected to the computer for printing of payment bill.

On the other hand, Graphical User Interface (GUI) was developed by involving Tkinter to display the detected products along with their price. Concurrently, the GUI consisted of several buttons such as start, checkout, delete, print and exit to ease the interaction between user with the checkout system.

## CHAPTER 3

### METHODOLOGY AND WORK PLAN

#### 3.1 Introduction

This chapter will include a detailed methodology in developing a software prototype of cashierless checkout system with product recognition as well as price computation functionality. First of all, overall system architecture of the proposed computer vision-based checkout system will be introduced followed by the workplan in accomplishing the goal. After that, each process will be described in depth, starting with preliminary model benchmarking, data acquisition and pre-processing, image synthesis and rendering as well as model optimization along with deployment on Jetson Nano which is a single-board edge device that supports model inferencing.

##### 3.1.1 Overall System Flow

The overall framework of our proposed computer vision-powered checkout system is made up of two main stages which are backend and frontend that play different roles as illustrated in Figure 3.1.

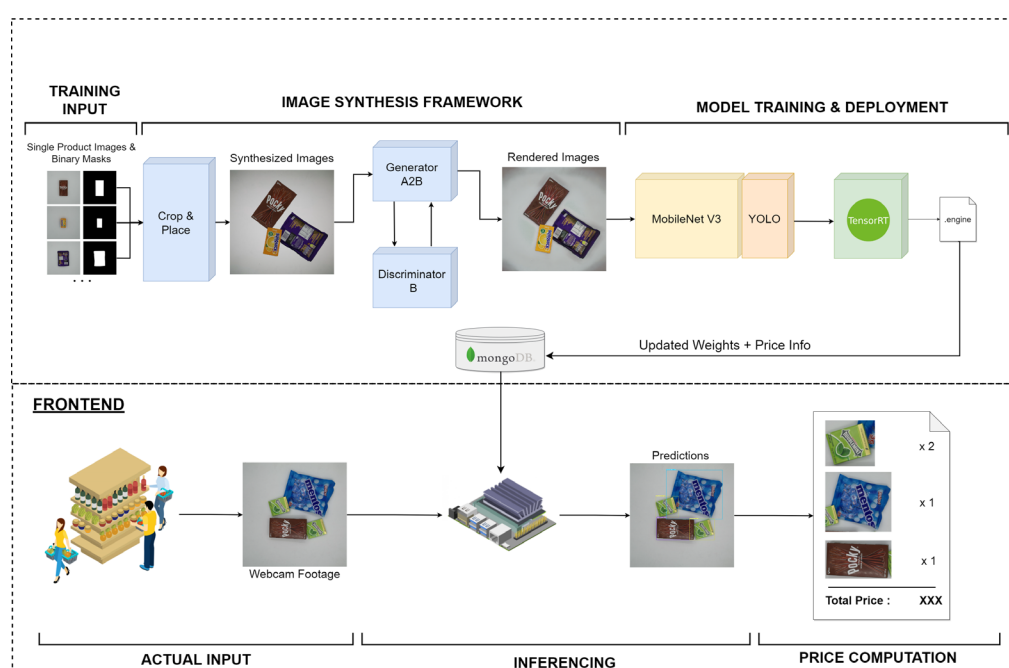


Figure 3.1: Proposed Cashierless Checkout System Architecture

In general, the backend framework acts as a platform that allows training and deployment of deep learning model to database. Firstly, the backend will involve acquisition process of training data where single product images and their corresponding binary masks will be prepared and serve as training input. Subsequently, the images will be fed to a custom image synthesis framework similar to the approach in the paper by Li et al. (2019) and Wei et al. (2019) where each of the images is pre-processed and rendered using a Generative Adversarial Network (GAN) to simulate actual scenario of a checkout counter, forming a reliable data for training. During the process, every instance in the image will be labelled automatically by the algorithm to minimize human workload in training a product recognition model. After that, training of an improved YOLOv5 will be carried out using the dataset before the updated weight to be uploaded to database after optimization by TensorRT runtime.

On the other hand, frontend of the proposed checkout system involves inference and calculation of overall product prices based on the items placed on retail store checkout counter by customer. The checkout process begins with the placement of desired products under the camera. The inference will then be executed using an edge device after all items are placed still on the counter. With the predicted output by the product recognition model, price of each item will be computed and displayed to customer for confirmation before proceeding to payment gateway. Concurrently, when there is any updated weight or product price available in the database, the edge device will fetch the latest model and price to ensure a reliable checkout procedure.

### **3.1.2 Work Plan**

Throughout this project, Waterfall methodology was adhered to ensure the completion of computer vision-based checkout system within this Final Year Project (FYP) period. According to Sherman (2014), Waterfall methodology is essentially a linear approach used in software development, where the tasks are split into phases and handled sequentially. This allows the understanding of project scope, schedule and workload at initial stage of the project before starting to move into development and technical part of the project. Additionally,

tracing of project status is simpler with a complete schedule as well as resource plan.

Figure 3.2 illustrates the waterfall diagram along with detailed tasks associated in each phase.

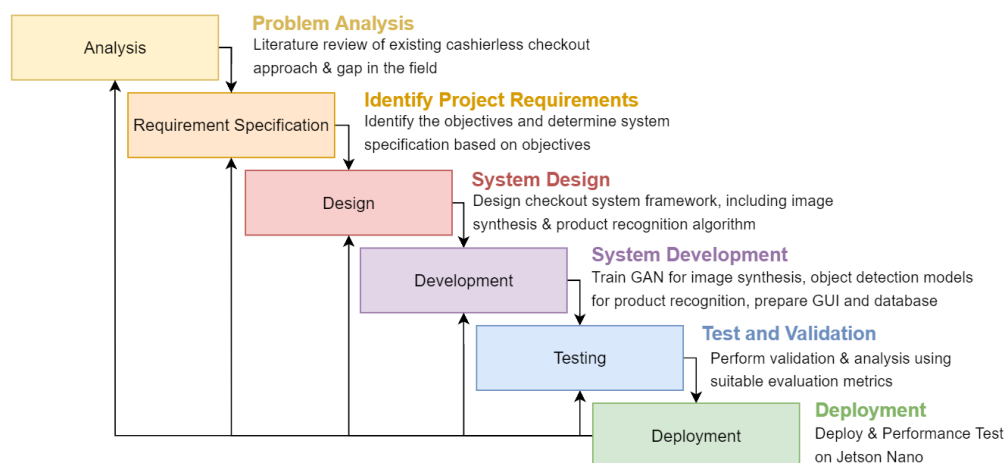


Figure 3.2: Waterfall Diagram for Project Prototype Development

### 3.2 Dataset Preparation

At the initial stage of the project, two datasets were being prepared before conducting any development and experiment of image synthesis algorithm as well as deep learning models where each of them was meant for different stage of training and evaluation.

As mentioned in Chapter 2.2.5, the first dataset was based on MVTec D2S Dataset as the publicly available dataset can provide a persuasive preliminary benchmark among different product recognition models commonly used in existing works because it fully represented product occlusion, variation of lighting and intraclass products at checkout counter. In addition, with consideration of time limit and hardware constraint, another small-scale dataset was prepared by adopting the concept of both D2S and RPC Dataset so that it can be used for training as well as evaluation of GAN-based image synthesis algorithm and product recognition models.

The steps taken in preparing both datasets will be explicitly described in the following subchapters.

### 3.2.1 MVTec D2S Dataset

Since the test set of MVTec D2S Dataset is incomplete, extraction was done in similar method in Ning, Li and Ramesh (2019) which only involved validation set because images in the subset can represent the characteristics of the entire dataset, including occlusion of products, lighting and intraclass variation. By constructing a modified python script, validation images were divided into 2 splits at a ratio of 8:1:1 under a random state of 12. As a result, an annotated dataset with 2880 training images, 360 validation and 360 testing images was formed where the image distribution was identical to the research by Ning, Li and Ramesh (2019). Samples of each split can be observed in Figure 3.3 (a) along with overall distribution of instances in (b).

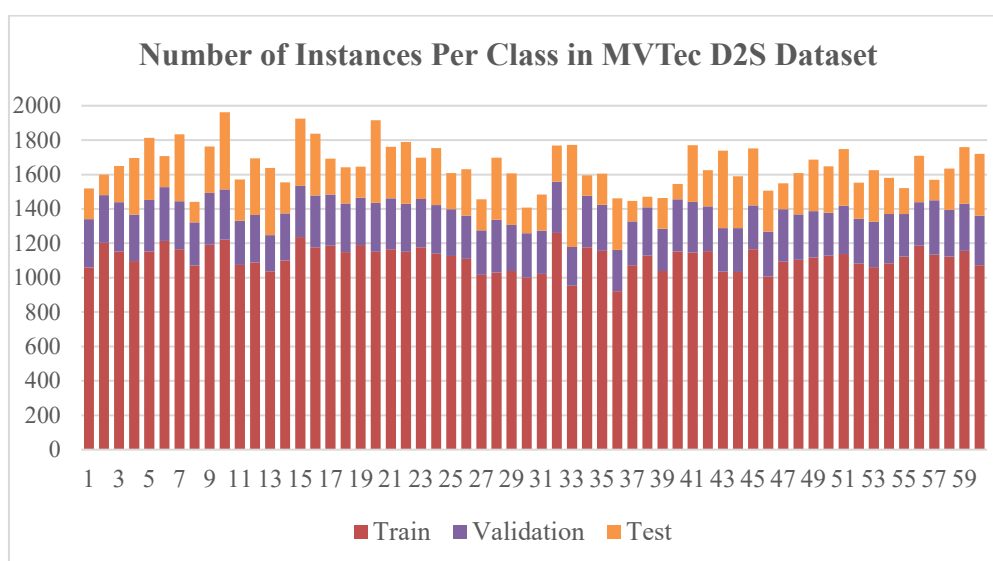
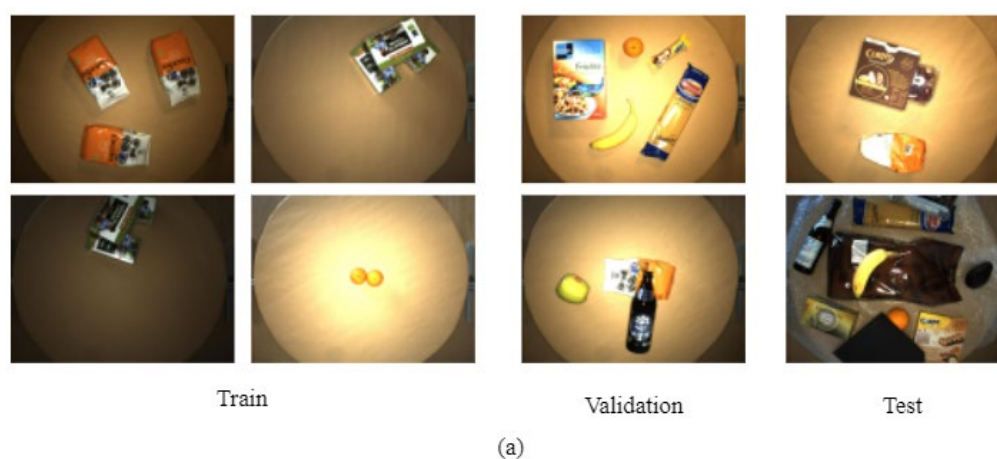


Figure 3.3: Samples (a) and Instances Per Class (b) in MVTec D2S Subset



After that, by utilizing the same python script and LabelImg by Dutta and Zisserman (2019), 3 new annotation files were generated from the provided validation JSON file (.json) in COCO format. However, in order to accommodate distinct annotation file format and image directory requirements between different deep learning models, the dataset was further processed according to file structures shown in Figure 3.4.

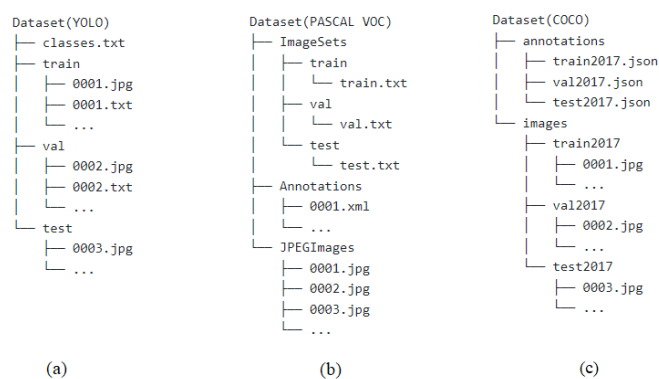


Figure 3.4: File Structure of YOLO, PASCAL VOC and COCO Dataset

In YOLO format, bounding boxes information were stored as normalized values of x, y center coordinate along with their normalized width and height in text files (.txt) named according to image file name. Additionally, all product classes were extracted and stored in *classes.txt*. With all the files being generated, all images and their annotations were added into *train*, *val* and *test* folders.

As for PASCAL VOC, bounding boxes were stored as non-normalized coordinates of diagonal corners in Extensible Markup Language (XML) format (.xml) along with image name, file path and image size as well as object class name. After that, 3 text files that indicate the split of dataset were prepared through python script before placing images, annotation XML files and their split indication text files to corresponding folders named as *ImageSets*, *Annotations*, and *JPEGImages*.

Concurrently, bounding boxes were stored in as minimum values of x and y coordinates but along with their box width and height in a single JSON format (.json) along with other information like object classes, image size and

name. Subsequently, all images were placed into corresponding child folders in *images* while annotations were placed in *annotations* folder.

### 3.2.2 Custom Dataset Preparation

Additionally, another small-scale dataset was prepared from scratch for the training of GAN-based image synthesis framework and subsequent development of product recognition algorithm. The dataset consisted of 20 different classes of groceries that can be commonly found at retail store in Malaysia. The products were selected in such a way that they will be distinct in sizes, colors, and shapes in order to have a better representation of product range available in common retail stores.

#### 3.2.2.1 Image Acquisition Setup

A controlled image acquisition setup is crucial in preparing a dataset for any deep learning models because uncontrolled setup will cause the images to be susceptible to distortions like blur and noises which will reduce the performance of deep learning models (Dodge and Karam, 2016). Thus, a controlled image acquisition environment was set up by involving a portable photo booth with plain white background and uniform lighting as shown in Figure 3.5:



Figure 3.5: Image Acquisition Setup

By using a smartphone camera, each product was placed into the booth before its front side and back side were captured at a resolution of  $2976 \times 2976$ , forming a training subset with 37 images in total. As for validation set, 80

images were captured where each image was made to consist of 2 to 3 classes of products that were randomly placed in the booth to imitate the actual checkout counter scenario. Furthermore, test set was separated into scenarios with overlapping and without occluded products to assess the effectiveness of image synthesis algorithm in handling them, forming 40 images for each scenario. Some samples in the raw dataset can be seen in Figure 3.6.



Figure 3.6: Samples in Raw Dataset

### 3.2.2.2 Annotation

After obtaining the raw dataset, annotation process of images was carried out. Based on the definition by Potter (2021), annotation is about addition of metadata to a dataset so that computer vision models can identify and differentiate the objects during training and provide accurate predictions. Concurrently, labelled images also serve as ground truth for evaluating the model's performance. In this project, 2 types of image annotations were involved and each of them was used for image synthesis algorithm and product recognition algorithm respectively.

Firstly, segmentation mask was prepared by marking the boundary of every individual product in each image using another online tool known as VGG Image Annotator (VIA) released by Dutta and Zisserman (2019) shown in Figure 3.7. When all the images were annotated, JSON file (.json) with COCO formatting was exported.

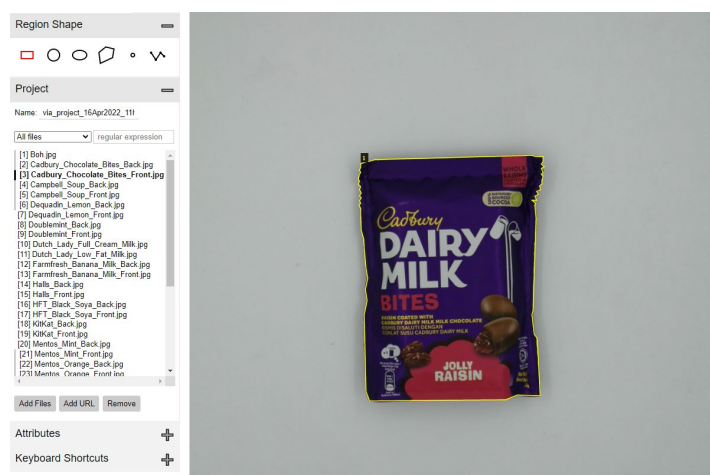


Figure 3.7: VGG Image Annotator

On the other hand, second type of image annotation which involved labelling of product classes and location was done by utilizing same LabelImg by Tzutalin (2015) mentioned in Chapter 3.2.1. Similarly, the dataset adhered 3 different annotation and file directory format so that they can be used for different deep learning model architectures.

### 3.3 Development of Image Synthesis Framework

In order to simulate the actual checkout condition with occluded products, intraclass variation and lighting differences, a novel image synthesis framework was constructed based on the approach in Li et al. (2019), Rigner (2019) and Wei et al. (2019). The framework was comprised of 3 main modules, starting with binary mask extraction, crop and place algorithm, GAN model for shadow synthesis, and image augmentation to include lighting variation. Each of them will be described in detail in following subchapters.

#### 3.3.1 Binary Mask Extraction

Extraction of binary mask played a crucial role in the GAN-based image synthesis algorithm because products will need to be cropped according to their packaging before they can be used to generate synthetic image that simulates randomly placed products at checkout counter.

To achieved this, a python script was constructed. Firstly, it will take user input about the image directory and segmentation annotation in JSON format (.json) generated from VIA annotator in Chapter 3.2.2.2. From the

annotation file, each point used to form segmented mask will be extracted. Subsequently, binary mask can be created using `cv2.fillPoly()` function on a blank, black image with the same resolution as raw image ( $2976 \times 2976$ ). Generated binary mask will be as illustrated in Figure 3.8.

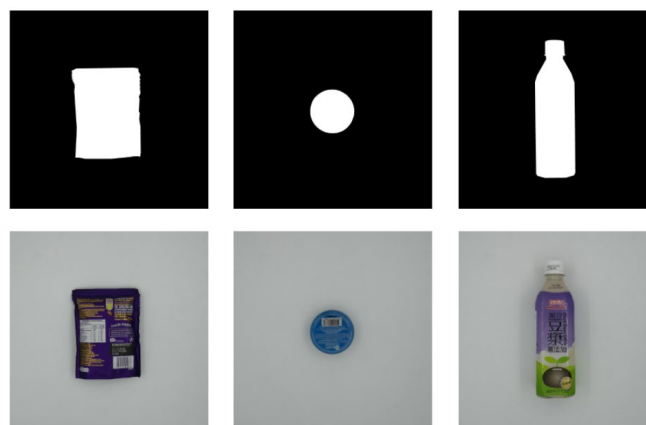


Figure 3.8: Binary Masks of Retail Products

### 3.3.2 Crop and Place Algorithm

Crop and Place Algorithm used in this project was developed based on the implementation in Li et al. (2019). Firstly, by specifying number of pictures to be generated and annotation file in JSON format (.json) to the script in Appendix A, the algorithm will randomly pick 3 product classes to be included in the synthetic image. For each selected product class, algorithm will read the corresponding raw image and its mask before performing random rotation and cropping of the product's ROI. Then, instead of direct pasting of product onto an empty checkout counter background, ROI will be used to calculate the Intersection over Union (IoU) with existing objects placed in the image through Equation 3.1 so that level of product occlusion can be controlled, and smaller products will not be fully covered up.

$$IoU = \frac{A \cap B}{A \cup B} \quad (3.1)$$

where:

$A$  = ROI of current product

$B$  = ROI of existing products in background image

After the computation of IoU, product will be masked and pasted into the empty checkout counter background using the function, *Image.paste()*. Besides, information of bounding boxes and product classes will be written to new COCO formatted JSON file (.json). Ultimately, a series of fully annotated, synthesized training images that can simulate random placement of products on a checkout counter as in Figure 3.9 with minimal human involvement.



Figure 3.9: Synthesized Training Images

### 3.3.3 GAN-based Shadow Synthesis

In order to construct a realistic training data that can imitate the real checkout situation without requiring additional human involvement, shadow synthesis was done using GAN similar to the approach Li et al. (2019) and Wei et al. (2019). In this project, two state-of-the-art GAN(s) known as CycleGAN and AttentionGAN were constructed based on their official repositories as in Appendix E. Each of them was trained under the same hyperparameters to allow a fair comparison. The model with higher performance will be used to construct a reliable training set for product recognition models. The parameters that were used are summarized in Table 3.1.

Table 3.1: Hyperparameters for GAN

Hyperparameters	Description	Value
<b>netG</b>	Generator type	ResNet-9
<b>n_epochs</b>	Epoch number (No decay)	100
<b>n_epochs_decay</b>	Epoch number (decay)	100
<b>batch_size</b>	Batch size	4
<b>preprocess</b>	Training image preprocess	scale_width_and_crop
<b>load_size</b>	image size	800

Table 3.1 (Continued)

<b>crop_size</b>	<b>cropped image size</b>	<b>256</b>
<b>lambda_identity</b>	Identity mapping loss scale	0.4
<b>lambda_A</b>	Cycle loss weight (A to B)	8
<b>lambda_B</b>	Cycle loss weight (B to A)	8

### 3.3.3.1 CycleGAN

CycleGAN represents an extension of GAN released by Zhu et al. (2020) that utilizes the concept of 2 models in GAN known as generator and discriminator. A generator plays the role in generating fake images while a discriminator will evaluate the samples' probability to differentiate between generated and real images.

In terms of architecture, CycleGAN employs 2 pairs of generator and discriminator models for translation of images from their corresponding domain as illustrated in Figure 3.10. Generator A will take input from domain A to generate images in domain B before passing to discriminator A for evaluation. Oppositely, generator B will take input from domain B to generate images in domain A and evaluation is done through discriminator B. Probabilities calculated through both discriminators will be used to update generator models to reduce dissimilarity between images.

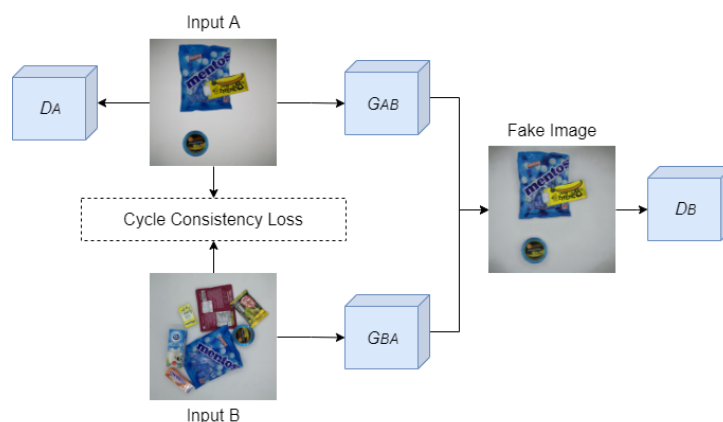


Figure 3.10: Architecture of CycleGAN

### 3.3.3.2 AttentionGAN

AttentionGAN published by Tang et al. (2021) is a GAN variant that was meant for translation task of unpaired images similar to CycleGAN. However, as the

name suggests, AttentionGAN included attention mechanism so that the generators can effectively differentiate the foreground and background objects, allowing a conservative translation between two image domains where the change of background can be minimized. This feature is preferable since image synthesis algorithm involved in this project will only require rendering of shadow without affecting the products in the image.

The architecture of AttentionGAN used in this project represents the second scheme introduced in the paper that comes with two generators with built-in attention layers. As shown in Figure 3.11, input image will be fed to one of the generators,  $G$  where its Content Mask Generator,  $G_c$  will generate a content mask,  $C^f$  that represents the transformed foreground object while Attention Mask Generator,  $G_A$  will create Foreground,  $A^f$  and Background Attention Masks,  $A^b$  to differentiate them. Subsequently, both content masks and Foreground Attention Masks,  $A^f$  will be multiplied to mask out domain transformation of background. Concurrently, Background Attention Masks,  $A^b$  is multiplied with input image to obtain another intermediate image preserved background. Finally, by fusing both intermediate images, a realistic transformed image can be formed with minimal background alteration. After that, the image will be fed to another generator with identical structure to revert the transformation so that Cycle Consistency Loss can be computed and updated to generators.

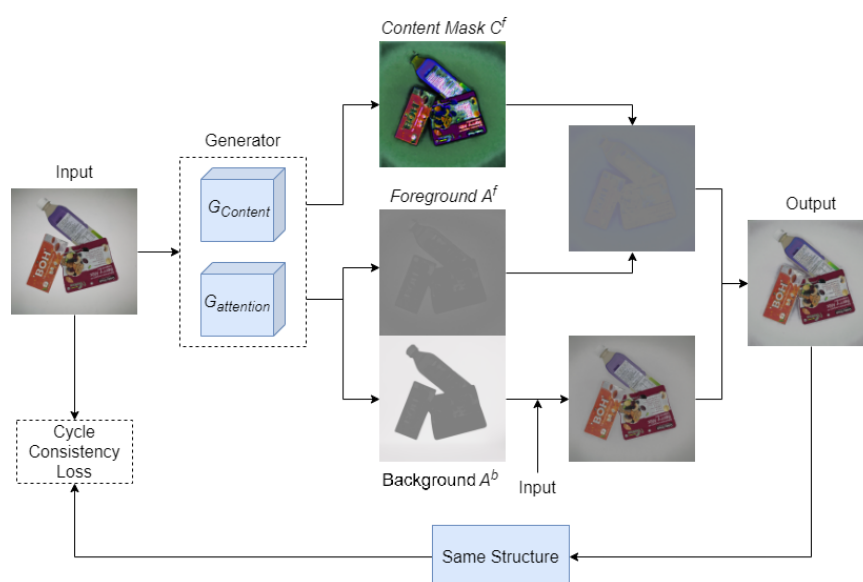


Figure 3.11: Architecture of AttentionGAN



### 3.3.4 Lighting Variation

After applying shadow synthesis using GAN, lighting variation was added to each rendered image using conventional data augmentation approach similar to Rigner (2019) to further extend the reliability of training dataset. In this project, Albumentations data augmentation library by Buslaev et al. (2020) was used as it encompasses vast amount of image augmentation operation while providing support to bounding box augmentation.

As attached in Appendix B, the python script will first read user input about number of images to be generated per input image, as well as directory of images and annotations. After that, for each input image in the directory, multiple image and bounding box augmentation will be performed using *Albumentation.Compose()* and *Albumentation.transform()* before each of the augmented image and bounding boxes were written to a new directory. All augmentation used in this project can be summarized in Table 3.2.

Table 3.2: Configuration of Data Augmentation

Function	Parameters	Value
<b>RandomRotate90()</b>	<b>probability</b>	1.0
<b>Resize()</b>	<b>height</b>	800
	<b>width</b>	800
	<b>interpolation</b>	cv2.INTER_AREA
<b>ShiftScaleRotate()</b>	<b>rotate_limit (°)</b>	5
	<b>border_mode</b>	cv2.BORDER_REFLECT_101
	<b>probability</b>	1.0
<b>ISONoise</b>	<b>color_shift</b>	(0.01, 0.05)
	<b>intensity</b>	0.1, 0.5
	<b>probability</b>	0.5
<b>RandomBrightness-Contrast()</b>	<b>brightness_limit</b>	0.2
	<b>contrast_limit</b>	0.2
	<b>probability</b>	0.5

Ultimately, the self-prepared dataset was expanded up to 3000 training images, 600 of validation images, 600 test images of overlapped products and 600 images of non-overlapped products shown in Figure 3.12 followed by its class distribution as in Figure 3.13.



Figure 3.12: Samples in Expanded Dataset

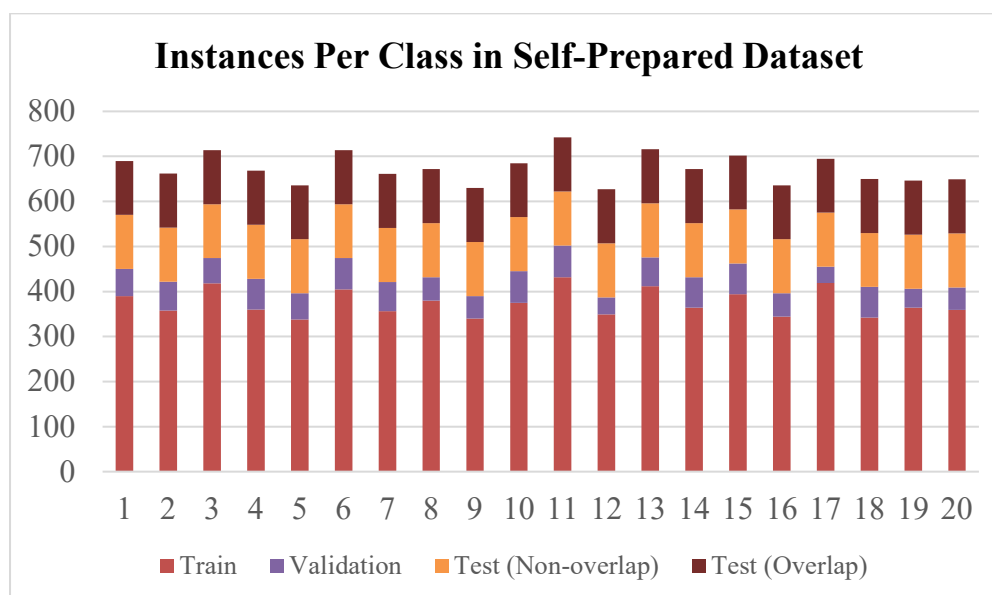


Figure 3.13: Instances Per Class in self-prepared dataset

### 3.4 Selection of Baseline Object Detection Model

As reviewed in Chapter 2.4.2, several types of models were used in the field, including R-CNN series under two-stage detector category, YOLO models as

well as RetinaNet under single-stage detector. However, instead of choosing a baseline model randomly for improvement, filtering will be required through preliminary benchmarking of the models.

Therefore, model architectures were analysed but single-stage models were prioritized because deep learning-based retail product checkout asserts high requirement on the accuracy and inference speed of the object detection model as checkout process needs to be performed rapidly with adequate accuracy (Xie, Wang and Zhao, 2021). Subsequently, benchmarking was carried out to assess their actual performance in retail product recognition.

### **3.4.1 State-of-the-Art Model Architectures**

#### **3.4.1.1 YOLOv3**

As stated by Redmon and Farhadi (2018), YOLOv3 represents an updated algorithm based on YOLOv2 by updating the initial backbone known as DarkNet-19 to a deeper structure named as DarkNet-53. This is because YOLOv2 with DarkNet-19 is only comprised of 19 convolutional layers and 11 additional layers to perform object detection and continuous downsampling further causes loss of fine-grained features and result in poor performance especially in detecting small objects. In contrast, YOLOv3 improvised the backbone network to 53 convolutional layers and included shortcut layers that resembles the skip connection in ResNet which will add the output from previous layer to subsequent layer. As claimed by Mantripragada (2020), this configuration will aid the training of deep networks without resulting diminishing gradient.

After the backbone structure, YOLOv3 architecture is followed by multi-scale detection head through another 53 convolutional layers. As described by Redmon and Farhadi (2018), their algorithm performs detection at three different scales by first downsampling feature maps by 3 different ratios which are 8, 16, 32 for detection of small, medium and large object respectively before each of them are pass for detection at respective convolutional layer with 1 x 1 filter. For detection of large object, it was done by taking feature map that is downsampled by ratio of 32 in previous layers and passed to 82<sup>nd</sup> layer for prediction. Subsequently, the previous feature map will be upsampled by 2 before detection of medium objects at 94<sup>th</sup> layer. Lastly, the feature map size is

further increased by ratio of 2 for small object detection at 106<sup>th</sup> convolutional layer. Ultimately, the structure allows YOLOv3's performance to surpass YOLOv2 especially in detection of small-sized objects. Overall architecture can be represented in Figure 3.14.

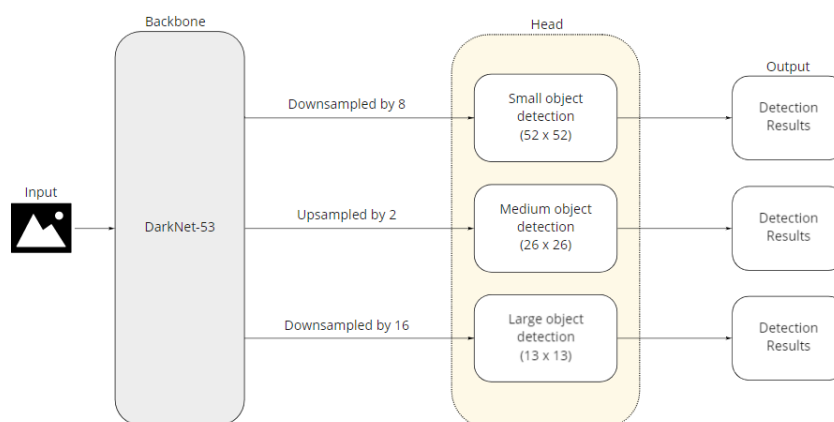


Figure 3.14: YOLOv3 Model Architecture

### 3.4.1.2 YOLOv5

As the name suggests, YOLOv5 represents the fifth architecture under YOLO family that is written in Python language rather than C language adopted in previous versions of YOLO. However, similar to YOLOv4 that was released by Bochkovski, Wang and Liao (2020) in the same year due to parallel commencement of research, YOLOv5 achieved similar results compared to YOLOv4 but greatly shorten the training and inference time.

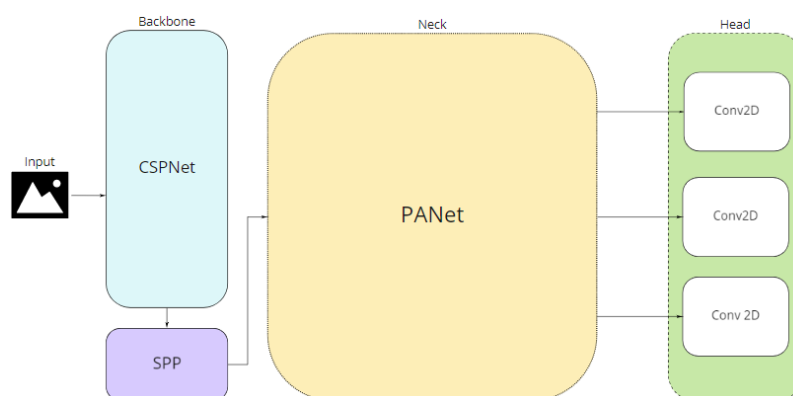


Figure 3.15: YOLOv5 Architecture

As shown in Figure 3.15, YOLOv5 can be decomposed into 3 modules, namely backbone, detection neck and head. The backbone of the model is

composed of Cross Stage Partial Network (CSPNet) and Spatial Pyramid Pooling (SPP) Layer. In CSPNet, the feature map will be directed into two paths where one of them involves a dense block and a transition layer. The feature map will pass through both paths and merge together for the next layer. This configuration is claimed to reduce repeated gradient during training while maintaining the model's complexity (Wang et al., 2019). As for SPP, variable-sized feature maps can be pooled with multiple kernel sizes ( $5 \times 5$ ,  $9 \times 9$ , and  $13 \times 13$ ) to generate a fixed-sized output, making the backbone to be scale-invariant (Jocher et al., 2021).

In detection neck, Path Aggregation Network (PANet) is used. It is an enhanced Feature Pyramid Network (FPN) that introduces lateral connection between a bottom-up and top-down pyramid structure that speeds up information flow. Besides, it also includes adaptive feature pooling to extract information from each feature level before each of the aligned feature map is combined for enhanced object localization. As for the detection head, YOLOv5 will predict at different scales ( $18 \times 18$ ,  $36 \times 36$ ,  $72 \times 72$ ) similar to YOLOv3, making it robust in handling objects of multiple sizes (Xu et al., 2021).

Additionally, as stated by Thuan (2021), YOLOv5 also introduces auto learning anchors that can effectively compute the most suitable anchor sizes for any custom dataset through K-means and genetic learning algorithms. The best-fit anchor boxes will help to the model to converge faster, thus improving training time and accuracy. Other than that, YOLOv5 is flexible by offering 4 models with different model complexity, namely YOLOv5s, YOLOv5m, YOLOv5L and YOLOv5x. Each of them targets different computing platform from edge devices to cloud deployment.

### 3.4.1.3 RetinaNet

RetinaNet represents a network that was designed to tackle the issue of low foreground-background ratio which is commonly found in single-stage object detectors. Hence, by proposing a new loss function in the existing single-stage architecture, RetinaNet is able to achieve high accuracy as in two-stage object detectors while attaining the simple structure and high speed of single-stage models. According to Lin et al. (2018), the loss function can be referred as Focal Loss which is based on Cross Entropy Loss. As shown in Equation 3.2, when

focusing factor  $\gamma$  is larger than 1, misclassified object will have a low  $p_t$  probability value, hence increasing the coefficient to 1, leaving the weight unaffected. Oppositely, weight will be downscaled to 0 for well-classified object due to high  $p_t$  value. As a result, the network will emphasize on foreground objects that are hard to detect through larger weights while reducing the importance of easy examples like background.

$$\text{Focal Loss } (p_t) = -(1 - p_t)^\gamma \log(p_t) \quad (3.2)$$

In terms of architecture, RetinaNet can be dismantled into three main components which consist of a backbone network and two sub-networks for object classification and bounding box regression. At the beginning of the network, a custom structure which involves combination of ResNet and FPN are adopted. ResNet serves as a bottom-up pathway to efficiently generate feature maps at different scales regardless of input image size. Subsequently, the backbone is followed by FPN that adds a top-down pathway and form lateral connection with ResNet. The FPN is composed of 5 different levels ( $P_3$  to  $P_7$ ) and 256 channels to provide a rich feature map that is scale-invariant and boost the speed and accuracy of the model. After that, two parallel and identical subnetworks are connected to each FPN level as subnetworks. Both of them are made of  $3 \times 3$  convolutional layers with 256 filters and end with another  $3 \times 3$  convolutional layer with the filters that varies in number according to object classification or bounding box regression task (Deshmukh, 2020). The architecture of RetinaNet can be summarized as in Figure 3.16.

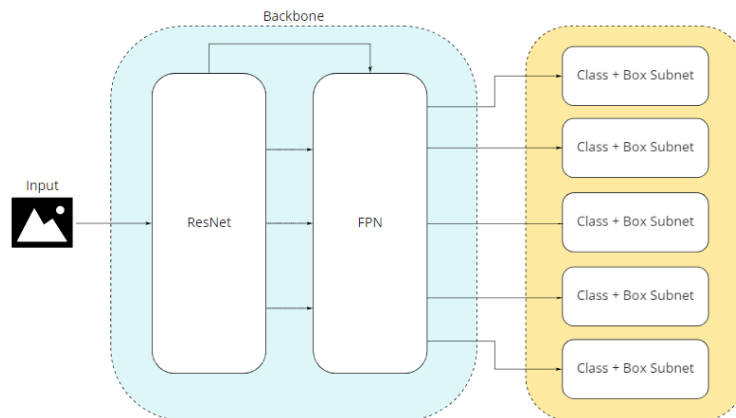


Figure 3.16: Architecture of RetinaNet

### 3.4.2 Preliminary Benchmarking

By utilizing MVTec D2S Dataset prepared in Chapter 3.2.1, preliminary benchmarking was carried out to justify their actual product recognition performance. Similar to the approach by Bukhari et al. (2021), training of model was carried out in a web IDE released by Google ( n.d.) known as Google Colaboratory due to the reason that it offers additional computing power through professional graphics card such as NVIDIA Tesla K80 to NVIDIA Tesla T4 with large VRAM to speed up training of deep learning algorithms under python environment. To ensure that the results can be compared under same fair ground between all models, training was done with the same GPU, which is NVIDIA Tesla T4 equipped with 16GB of VRAM as illustrated in Figure 3.17.

```

NVIDIA-SMI 460.32.03      Driver Version: 460.32.03      CUDA Version: 11.2
-----+-----+-----+-----+-----+-----+-----+-----+
GPU  Name          Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC
Fan  Temp   Perf   Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M.
                                           | MIG M.
-----+-----+-----+-----+-----+-----+-----+
 0   Tesla T4             Off | 00000000:00:04.0 Off |                    0
N/A   37C    P8     9W / 70W           0MiB / 15109MiB      0%      Default
  
```

Figure 3.17: GPU Specification in Google Colaboratory

Prior to training process, python codes for each model were constructed as attached in Appendix C, D, F by referring to implementation on each official repository as well as other repositories that allowed easier implementation based on Pytorch framework due to higher flexibility and friendly for developers supported by the statement of Dubovikov (2018). At the same time, to ensure each model can be compared on the same grounds, hyperparameters were configured as in Table 3.3 and remained constant throughout the training of each model. Additionally, several model-specific parameters were set due to difference in architectures.

Table 3.3: Hyperparameters for Product Recognition Model

Hyperparameters	Value
Epoch / *Max Batches	150 / 27000
Classes / *Filter	60 / 195

Table 3.3 (Continued)

<b>Batch Size / *Subdivision</b>	16 / (4 × 4)
<b>Input size</b>	512 × 512
<b>Learning rate</b>	0.001
<b>Optimizer</b>	Adam

\* YOLOv3-specific hyperparameters

### 3.5 Model Optimization

Based on the results of preliminary benchmarking, YOLOv5 was selected as a baseline model, and it was further optimized and fine-tuned to enhance its performance on edge devices with limited computational power which is Jetson Nano. In order to achieve this, backbone architecture of YOLOv5 which is CSPNet mentioned in Section 3.4.1.2 will be substituted with three different light-weight CNN architectures to reduce computational load so that retail product recognition can be carried out on edge devices with the best trade-off between accuracy and inference speed.

Three backbone structures involved in this project were the latest representative models from their family, which encompassed MobileNet V3, ShuffleNet V2 and GhostNet. Each of the backbone were studied in terms of their architectures and summarized in following sections before applying them to YOLOv5 models as shown in Appendix G. After that, by using self-prepared dataset in Chapter 3.2.2, each model was trained and evaluated. Additionally, other components of YOLOv5 model were remained unchanged such as hyperparameters were remained unchanged to ensure fair comparison.

#### 3.5.1 ShuffleNet V2

ShuffleNet V2 represents another light-weight CNN introduced by Ma et al. (2018) aimed to optimize speeds and Memory Access Cost (MAC) instead of FLOPs. Hence, the model was constructed based on ShuffleNet V1 and adheres 4 rules of efficient CNN architecture. First rule asserts that the network should possess same numbers of input and output channel to minimize the processing time per batch. Additionally, group convolution should be avoided as it increases MAC despite with the same FLOPs. Moreover, the network should



have less multi-path structure because it increases efficiency but greatly alters the overall computing efficiency. Concurrently, the network should utilize less element-wise operations such as Rectified Linear Unit (ReLU) as it requires larger MAC.

By complying to the 4 guidelines, ShuffleNet V2 possesses the architecture in Figure 3.18. Firstly, the input feature map will be split to two paths to avoid group convolutions. Both branches consisted of at least one  $1 \times 1$  convolutional layer and  $3 \times 3$  depthwise convolutional layer. After that, feature maps will be concatenated to form output feature maps with the same channel as input. Besides, unlike ShuffleNet V1, subsequent element wise operations were removed while preserving channel shuffle to allow information sharing between channel groups to reduce computational load while improving accuracy.

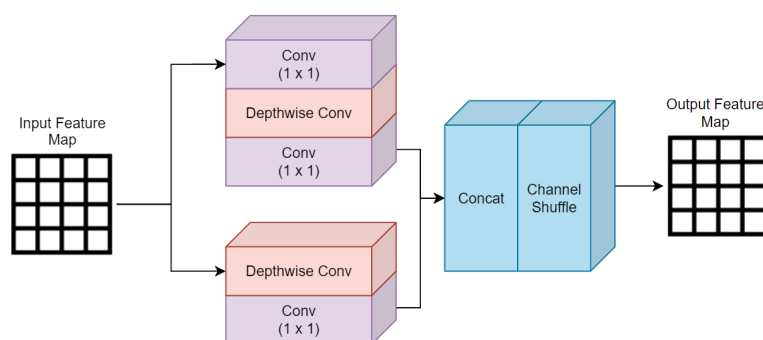


Figure 3.18: ShuffleNet V2 Architecture

Despite ShuffleNet V2 comes with several scales, the lightest variant (x1.0) was chosen to minimize the model complexity for deployment on Jetson Nano. Figure 3.19 demonstrates the modified architecture of YOLOv5 with ShuffleNet V2.

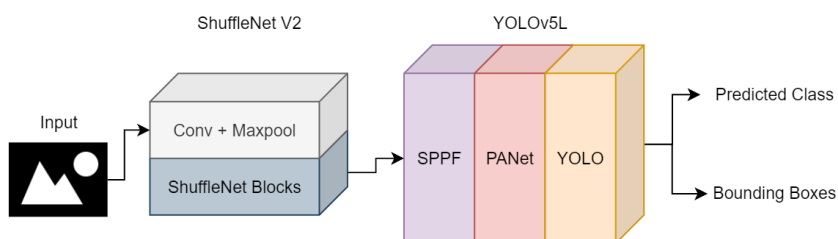


Figure 3.19: Proposed YOLOv5 with ShuffleNet V2 Backbone

### 3.5.2 MobileNet V3

MobileNet V3 represents the third version of MobileNet series which is a light-weight CNN designed to uplift the performance of embedded systems in carrying out model inferencing. As stated by Howard et al. (2019), MobileNet V3 inherits the concept of MobileNet family with Depthwise Convolutional Filters and Pointwise Convolution. As shown in Figure 3.20, in contrast to traditional convolutional filters that directly applies kernel with same depth as input image  $[W_1 \times H_1 \times N_1]$  to obtain an output, depthwise convolutional filters utilizes kernel with depth of 1  $[K \times K \times 1]$  and iterates through single channel of image. After stacking up 3 channels of output, an intermediate representation with size of  $[W' \times H' \times N_1]$  can be formed. Subsequently, pointwise convolution is applied by iterating a kernel with the size of  $[1 \times 1 \times N_1]$  through the intermediate image. As a result, an output of  $[W_2 \times H_2 \times N_2]$  is obtained. This implementation helps to reduce computational complexity since less multiplications are involved during the process.

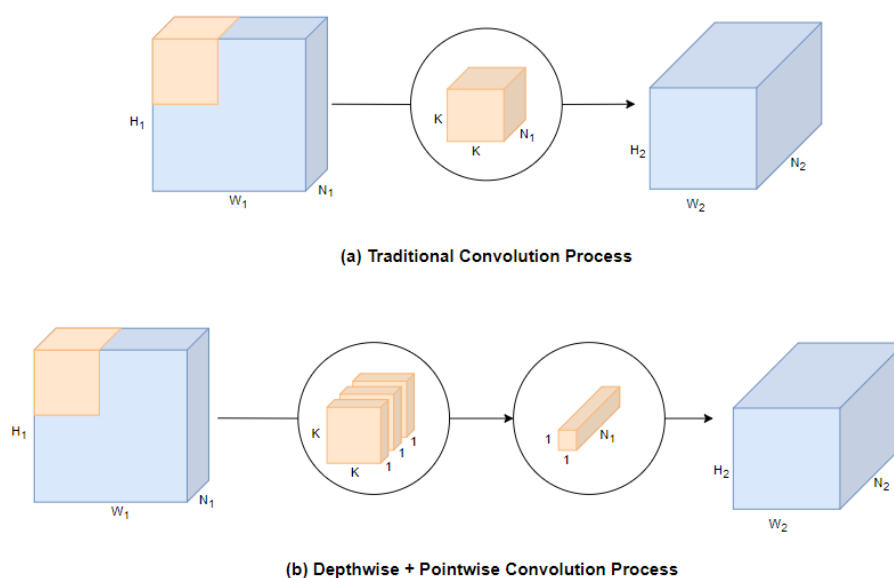


Figure 3.20: Depthwise and Pointwise Convolution Process

Furthermore, similar to its previous version, MobileNet V3 adopted Inverted Residual Block to preserve useful information under low feature dimension in such light-weight models. It was achieved by expanding feature dimension through pointwise convolution before passing the feature map to depthwise convolution layers. Besides, MobileNet V3 also implemented

Squeeze-and-Excitation (SE) layers which will compress and restore feature maps to emphasize important features before feeding to subsequent layers. This helps to increase accuracy while maintaining model size since SE layers are small and computationally cheap. As for activation function, MobileNet V3 adopts hard-Swish illustrated in Figure 3.21 that is faster to compute since no exponential function is involved compared to sigmoid loss function.

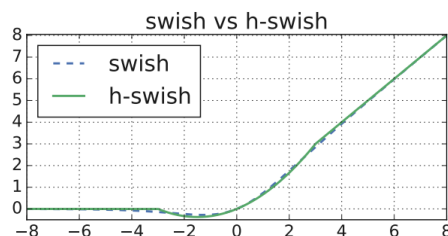


Figure 3.21: Hard-Swish Activation Function (Howard et al., 2019)

Despite there are two models available for MobileNet V3, smaller variant with 12 layers was chosen as the backbone of YOLOv5 as it is targeted for devices with limited computational resources will be used as backbone of YOLOv5 model to maximize the inference speed, as shown in Figure 3.22.

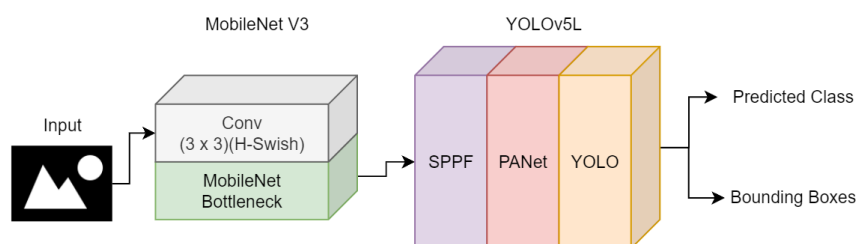


Figure 3.22: Proposed YOLOv5L with MobileNet V3 Backbone

### 3.5.3 GhostNet

GhostNet was developed by Han et al. (2020) with the aim to allow efficient deployment of CNN on devices with limited computation resources as well. In contrast to traditional CNN, GhostNet introduces a plug-and-play ghost module that is able to extract equivalent amount of feature maps during convolutional operations with lower Floating-Point Operations Per Second (FLOPS) because some of the feature maps will be similar and can be generated from other

essential feature maps using linear operations instead of using convolution that is computationally expensive.

As shown in Figure 3.23, the Ghost Module can be separated into 2 processes where the first part represents a conventional convolution with less channel to produce essential feature maps. After that, linear operation,  $\Phi$  is applied to each feature maps in order while identity mapping will be applied to original feature map for preservation. Through concatenation, an output feature maps can be formed.

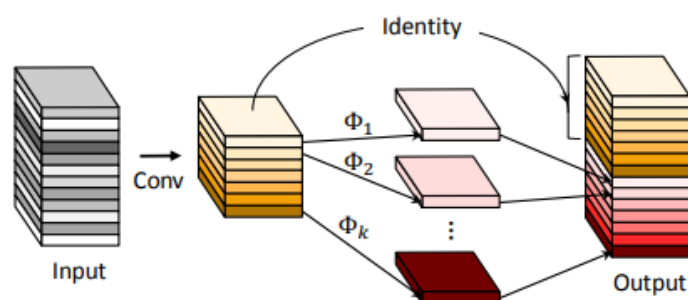


Figure 3.23: Ghost Module (Han et al., 2020)

After combining GhostNet with YOLOv5, the overall architecture will look as in Figure 3.24.

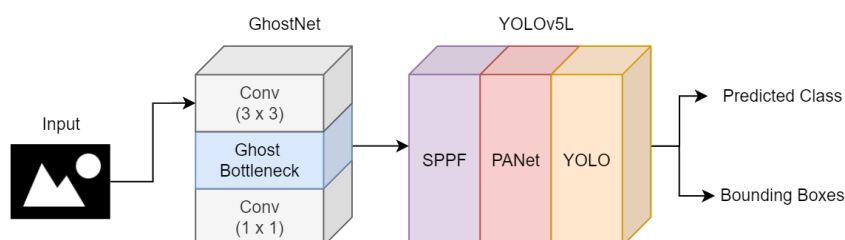


Figure 3.24: Proposed YOLOv5 with GhostNet Backbone

### 3.6 Computing Platform

In this project, the software prototype of cashierless checkout was deployed on a single-board computer developed by NVIDIA known as Jetson Nano, which is shown in Figure 3.25. It is because the device comes with several advantages compared to Raspberry Pi4 that is commonly used in the computer vision task. According to Table 3.4 that was constructed based on NVIDIA (2014) and

Raspberry Pi (2019), Jetson Nano is equipped with NVIDIA Tegra X1 with 128 Compute Unified Device Architecture (CUDA) cores that helps to accelerate inferencing performance compared to Broadcom Video Core VI that is meant for multimedia streaming. Additionally, Jetson Nano is able to provide 0.5 TFLOPS of computing power, which is 37% higher than Raspberry Pi 4 despite it requires an additional 1A of current.

Table 3.4: Raspberry Pi 4 and Jetson Nano Specifications

	<b>Raspberry Pi 4</b>	<b>Jetson Nano</b>
<b>FLOPS</b>	13.5 G	0.5 T
<b>CPU</b>	Quad-core ARM Cortex A72 @ 1.5 GHz	Quad-core ARM A57 @ 1.43 GHz
<b>GPU</b>	Broadcom Video Core VI (32-bit) @ 500 MHz	NVIDIA Tegra X1 w/ 128 CUDA cores @ 921 MHz
<b>Memory</b>	8GB LPDDR4	4GB LPDDR4
<b>Input Power</b>	5V 3A	5V 4A
<b>Camera</b>	MIPI CSI Port / USB	MIPI CSI Port / USB



Figure 3.25: Jetson Nano (NVIDIA, 2014)

### 3.6.1 TensorRT Acceleration

TensorRT is a runtime that provides optimization of deep learning models so that inference process can be accelerated on devices powered by NVIDIA GPU especially for those embedded systems with lower computing capability such as Jetson Nano. According to documentation by NVIDIA (2016), TensorRT involves 5 steps of optimization in maximizing the throughput of deep learning models on embedded systems. Firstly, models are quantized from Floating Point

32 (FP32) format to Floating Point (FP16) or Integer 8 (INT8) format. Through quantization, range of parameters can be reduced and result in smaller model weights. Additionally, model accuracy can be preserved by adopting symmetric quantization shown in Figure 3.26 where floating-point numbers in range are rounded and outliers are clipped to maximum of minimum value.

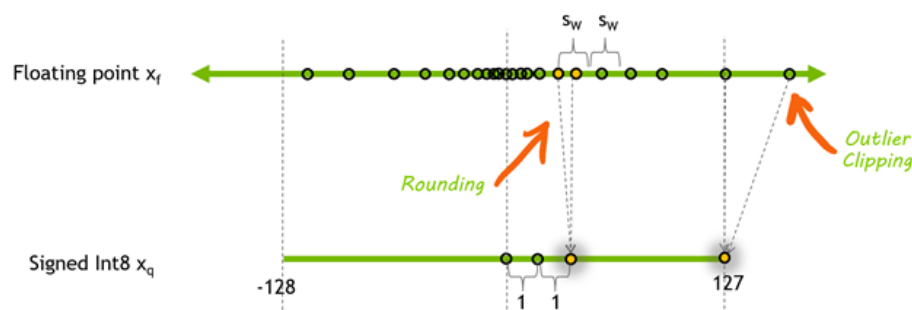


Figure 3.26: TensorRT Quantization (NVIDIA, 2016)

Furthermore, a deep learning model tends to have similar computation blocks that can unnecessarily occupy the GPU memory and reduce the overall efficiency. By using TensorRT, repetitive nodes can be fused together in horizontal or vertical manner so that lesser amount of tensor data will need to be read or write for each layer, making the model to be less memory intensive. In addition, TensorRT offer automatic tuning of model parameters based on kernel libraries of target platform to maintain the performance across different GPU. Lastly, TensorRT will utilize CUDA cores for parallel computation and dynamically allocate memory for each tensor for specific duration which will ultimately boost the performance on edge devices while minimizing memory usage as well as power consumption.

In this project, TensorRT was deployed to convert the proposed YOLOv5 model to Engine file format where the model weights are quantized to FP16 precision as it reduces the latency with minimal sacrifice of accuracy compared to INT8 that requires calibration dataset to preserve the model accuracy after quantization.

### 3.7 Evaluation Metrics

Several evaluation metrics were involved to analyse and compare between different GAN(s) and product recognition models that were experimented in this project. Each of the metrics will be described in the following subchapters.

#### 3.7.1 Fréchet inception distance (FID)

In this project, Fréchet inception distance (FID) was used to evaluate the effectiveness of GAN-based image synthesis framework along with qualitative analysis. FID introduced by Heusel et al. (2018) represents an evaluation metric that is commonly used to measure the performance of GAN(s) by comparing the similarity between real and generated images. When the generated image is having a low FID score, it is said to be similar to real images since feature vectors are closely correlated and less noise is present in the image.

Computation of FID Score involved a simplified Inception V3 model where its output layer is removed. Then, generated and real images will be fed to the model to generate feature vectors that represent them. After that, by utilizing both feature vectors, FID Score can be calculated using sum squared difference of two mean feature vectors and trace linear algebra,  $T_r$  in Equation 3.3.

$$FID = \|\mu_r - \mu_g\|^2 + T_r[\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2}] \quad (3.3)$$

Where:

$\mu_r$  = mean of real images (feature-wise)

$\mu_g$  = mean of fake images (feature-wise)

$\Sigma_r$  = Covariance matrix of real images

$\Sigma_g$  = Covariance matrix of generated images

#### 3.7.2 Mean Average Precision (mAP)

mAP represents an important metric used to evaluate the performance of an object detection model and it was widely adopted in Follmann et al. (2018), Rigner (2019) and Liu et al. (2020). However, calculation method of mAP can be divided into two approaches depending on type of generic object detection

challenge which are PASCAL VOC and COCO. Due to different calculation approach among them, PASCAL VOC was adopted for this project to ensure the fairness of comparison.

Before mAP can be calculated, precision,  $P$  and recall,  $R$  will need to be computed through Equation 3.4 and 3.5 by using three elements in confusion matrix which are True Positives (TP), False Positive (FP), and False Negative (FN). In this project, TP represents number of products that are correctly identified while FP indicates the number of misclassified product or background detected as product. As for the FN, it means the number of undetected products.

$$P = \frac{TP}{TP + FP} = \frac{TP}{\text{number of detection}} \quad (3.4)$$

$$R = \frac{TP}{TP + FN} = \frac{TP}{\text{number of ground truths}} \quad (3.5)$$

As summarized by Everingham and Winn (2012), mAP according to PASCAL VOC standard can be calculated by first obtaining all the recall values,  $R$  when there is a change of precision value,  $P$  exceeding the current maximum value through equation 3.4. Then, mAP can be calculated by obtaining the Area Under the Curve (AUC) of the interpolated Precision-Recall (PR) Curve using Equation 3.6.

$$P_{interp}(R_{n+1}) = \max_{R \geq R_{n+1}} (R) \quad (3.6)$$

### 3.7.3 Confusion Matrix

As claimed by Sammut and Webb (2017), Confusion Matrix can be defined as a metric used to evaluate the classification performance of a machine learning or deep learning model on a test set. As shown in Table 3.5, two axes of the matrix are occupied by true and predicted class where the former can be obtained from annotations while the latter is obtained from output of classifier. The first row demonstrates that all 10 objects in class A are correctly predicted while second demonstrates that there are two objects in class B that are misclassified as class A objects.



Table 3.5: Example of Confusion Matrix

		Predicted	
		A	B
Actual	A	10	0
	B	2	28

Similar to Bukhari et al. (2021), Confusion Matrix was also adopted in this project to evaluate the performance of product recognition model in predicting handling each product category including those with intraclass variation that can be easily misclassified.

### 3.7.4 Checkout Accuracy (cAcc)

cAcc represents an evaluation metric introduced by Wei et al. (2019) that was designed specifically for checkout process. It indicates the success rate of a product recognition algorithm and reflects the system's practicality in the actual checkout process because the metric will only consider the prediction as a success if and only if all products in the image are predicted accurately in the aspect of quantity and classes.

cAcc can be calculated by first obtaining prediction error for all product classes,  $CD_i$  using Equation 3.7, followed by Equation 3.8. When all products in an image is successfully predicted,  $CD_i$  will be zero, and  $\delta$  will provide output of 1 for cAcc computation.

$$CD_i = \sum_{k=1}^K |P_{i,k} - GT_{i,k}| \quad (3.7)$$

$$cAcc = \frac{\sum_{i=1}^N \delta(CD_i, 0)}{N} \quad (3.8)$$

Where:

$P_{i,k}$  = predicted count of k-th product class in i-th image

$GT_{i,k}$  = actual count of k-th product class in i-th image

$N$  = number of images in test set

### 3.7.5 Training and Inference Time

Training and inference time are crucial metrics because it indicates the practicality of a product recognition algorithm for the use in actual retail stores' checkout operation. Hence, training time was recorded for every model when their training of 150 epochs on Google Colaboratory was done. As for the inference time, the value represents the average time required for Jetson Nano to perform predictions of all 600 images of overlapped products in test subset.

## 3.8 Software Development

This project involved implementation of a python GUI that will make use of the developed product recognition algorithm to provide prediction of products and computation of its price. To achieve this, several components such as MongoDB Database service and Tkinter were used for the development.

### 3.8.1 MongoDB Database

MongoDB represents a scalable NoSQL database that is suitable for data with high volume. Unlike the conventional database service which make use of tables and rows, MongoDB utilizes key-value pairs known as documents as their basic data unit and sets of documents and function are contained in a structure known as collections, which is shown in Figure 3.27 along with the MongoDB admin portal.

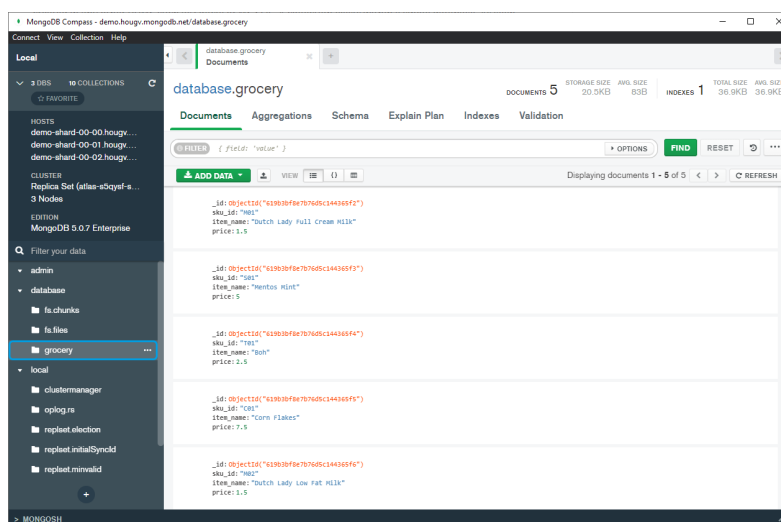


Figure 3.27: MongoDB Admin Portal

By including the python Application Programming Interface (API) provided by MongoDB, the proposed product recognition application will be able to fetch the latest model weight and product prices from the database to minimize human involvement in checkout system maintenance and updates.

### **3.8.2 Tkinter**

Tkinter represents an open-source built-in module that allows construction of GUI application in python. A Tkinter-based GUI application is essentially made up of a main window with interactive components known as widgets, which include buttons, labels, trees, entry. Their arrangement can be adjusted by specifying grid location in the window.

Tkinter is used in this project due to its portability and availability. According to Lutz (2006), Tkinter library is highly portable since it can run on Windows, MacOS and Linux without any modifying source code of the program. Additionally, Tkinter is already included in Python installation packages and can be utilized out of the box after installation of Python.

## CHAPTER 4

### RESULTS AND DISCUSSION

#### 4.1 Preliminary Benchmarking

At the initial stage of the project, several single-stage object detection models used in product recognition applications were being benchmarked using validation subset of MVTec D2S Dataset that is identical to Ning, Li and Ramesh (2019). The representative models involved in the evaluation process were YOLOv3, YOLOv5 and RetinaNet. Their performance was recorded and tabulated in Table 4.1 along with the two-stage Mask R-CNN used by the authors to serve as a reference.

##### 4.1.1 Quantitative Results

Table 4.1: Quantitative Performance of Representative Models

Model	Backbone Type	mAP (%)	Inference Time (ms)	Training Time (hrs)
<b>Mask R-CNN</b>				
(Ning, Li and Ramesh, 2019)	ResNet-50	(96.20)	75.00	-
<b>YOLOv3</b>	DarkNet-53	99.79	33.33	10.50
<b>YOLOv5L</b>	CSPNet	99.50	12.20	5.69
<b>RetinaNet</b>	ResNet-50	99.49	35.32	14.26

From the table, it can be observed that YOLOv3 achieved the highest mAP score of 99.79% while YOLOv5L and RetinaNet are close to each other in terms of their mAP with the value of 99.5% and 99.49% respectively. This result indicates that YOLOv3 can provide accurate detections across all 60 classes of retail products compared to YOLOv5L and RetinaNet. However, it can be inferred that all of the models can adapt to the actual checkout situation where

overlapping of products and variable lighting conditions are present during the inference process since all of the models achieved high mAP of above 99%.

Concurrently, in the perspective of average inference time, YOLOv5L has the shortest inference time of 12.20ms, which is 2.73 times faster than YOLOv3 with an inference time of 33.33ms. Meanwhile, RetinaNet requires 35.32ms for product detection which is slightly slower than YOLOv3 by 1.05 times and fall behind YOLOv5L by 2.895 times. This could be due to difference in terms of architecture where YOLOv5L has adopted CSPNet as its backbone which reuses gradients information from previous layers to update the weights during backpropagation, which asserts a lower computational cost compared to YOLOv3 and RetinaNet that utilize DarkNet-53 and ResNet-FPN as their backbone structure respectively. As for the training time, YOLOv5L has significantly low training time of 6 hours in contrast to YOLOv3 and RetinaNet which require 10.5 hours and 14 hours to complete a training session of 150 epochs.

Concurrently, it can be noticed that all three single-stage models have surpassed Mask R-CNN that was tested in Ning, Li and Ramesh, (2019) in terms of mAP despite the comparison may not be fair because the reference model was trained only for 10 epochs in their work. As for the average inference time, all models also clearly outperformed Mask R-CNN that requires 75ms to perform inference on 360 images in the test subset. This proves that single-stage models are far more efficient compared to two-stage architectures.

#### **4.1.2 Qualitative Results**

On the other hand, each model was tested in terms of their qualitative performance. It was done by choosing an unseen image to serve as constant variable for inference of all three models. The image chosen for the analysis was D2S\_068214 which comprised of 13 highly occluded products placed on checkout counter and all annotations were extracted from JSON annotation file (.json) provided. The ground truth image with labelled product names can be observed as in Figure 4.1 along with the images with predicted outputs generated through YOLOv3, YOLOv5L, and RetinaNet.

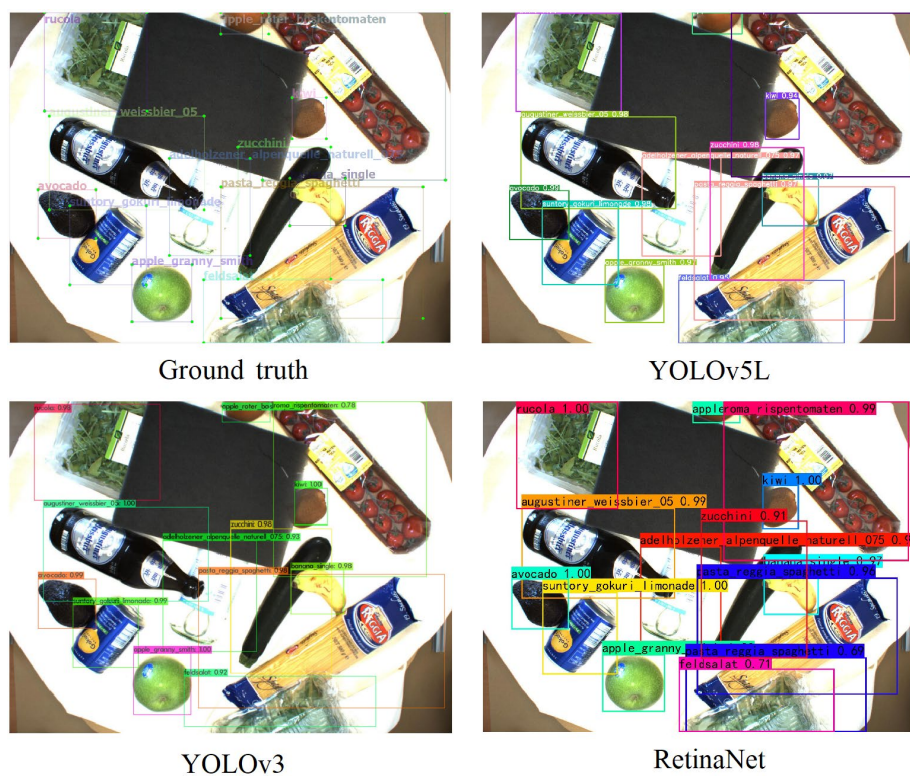


Figure 4.1: Ground truth and Predictions of Representative Models

Table 4.2: Confidence Score of Detected Products

Items	Confidence Score		
	YOLOv3	YOLOv5L	RetinaNet
adelholzener_alpenquelle_naturell_075	0.93	0.97	0.97
augustiner_weissbier_05	1.00	0.98	0.99
suntory_gokuri_limonade	0.99	0.98	1.00
apple_granny_smith	1.00	0.97	1.00
apple_roter_boskoop	0.89	0.95	0.95
avocado	0.99	0.99	1.00
banana_single	0.98	0.97	0.97
kiwi	1.00	0.94	1.00
pasta_reggia_spaghetti	0.98	0.97	0.96 (0.69)
feldsalat	0.92	0.95	0.71
roma_rispentomaten	0.78	0.97	0.99
rucola	0.98	0.98	1.00
zucchini	0.98	0.98	0.91

According to Figure 4.1, it is observed that each model has successfully detected all the products within the image and classified the correctly. However, there are some variations in terms of product localization and product confidence score between the models which can be seen through the bounding box location. Out of all detected products by YOLOv3, there are some bounding boxes that are misaligned compared to the ground truth images, especially for the long-shaped products like *pasta\_reggia\_spaghetti* and *roma\_rispentomaten*. In contrast, bounding boxes generated by YOLOv5 and RetinaNet are able to converge to the ground truth boxes closely compared to other representative models.

At the same time, according to confidence score summary in Table 4.1, it can be observed that RetinaNet and YOLOv3 achieved relatively maximum confidence of 1.0 for several distinct products such as *avocado*, *kiwi* and *apple\_granny\_smith* compared to YOLOv5. However, both models suffer from poor performance for products that are partially visible in the image. For instance, RetinaNet can only detect *rucola* with 0.71 confidence score while YOLOv3 only achieve a confidence score of 0.78 for *roma\_rispentomaten*. Moreover, RetinaNet generates a duplicate prediction of *pasta\_reggia\_spaghetti* on *feldsalat* due to occlusion between two products. This situation is similar with the research by Rigner (2019) and it is likely to be resulted by insufficient training of bounding box regressor in RetinaNet. In contrast, YOLOv5 detects the products with a uniform and high confidence because all scores are within 0.94 to 0.99 including those that are partially visible in the image.

In brief, it can be inferred that YOLOv5L can generalize better and outperforms YOLOv3 and RetinaNet in actual checkout situation with occluded products. Thus, it is more suitable for computer vision-based retail checkout applications and was used for subsequent development and analysis.





## **4.2 Effectiveness of Image Synthesis Framework**

### **4.2.1 GAN-synthesized Images**

In this section, two state-of-the-art GAN(s) which include AttentionGAN and CycleGAN, were being evaluated qualitatively as well as quantitatively to justify their effectiveness in simulating environmental features of real-world

checkout counter scenario. Qualitative analysis involves the observation and comparison of several samples generated by each GAN under same hyperparameters while quantitative analysis was done by comparing FID value between models. Both measurements are being tabulated in Table 4.3.

Table 4.3: Shadow Synthesis Results of AttentionGAN and CycleGAN

	Generated Samples				FID (↓)
<b>Input</b>					-
<b>Attention GAN</b>					46.82
<b>Cycle GAN</b>					40.99

By observing the generated samples, it can be noticed that both models are able to simulate shadows of the products to some extent. However, CycleGAN provides a smoother and realistic shadow compared to AttentionGAN especially when the product's packaging is irregular like *Nescafe* and *Mentos Mint*. Additionally, CycleGAN tends to cause less details degradation compared to AttentionGAN. It can be observed through the blurry product details from the images generated by AttentionGAN, but these details remained clear for the images rendered by CycleGAN. As for color degradation, CycleGAN tends to cause less color degradation in contrast to AttentionGAN for small-sized products. This can be observed through the color change of *Halls Black* from black color to green color as well as *Dequadin* that changes from dark blue to light blue in the leftmost image. Oppositely, when products occupy larger portion of the image, such as *Mentos Mint* and *Campbell Soup* in the third image, color degradation is more obvious for CycleGAN compared to



AttentionGAN. On the other hand, by looking at the background, it can be noted that AttentionGAN is more advantageous as the background is nearly identical to input image compared to CycleGAN that generates dark corners in the image. These phenomena are likely due to inclusion of attention mechanism in AttentionGAN where products will be focused as the foreground of the image and more conservative in dealing with the background. In contrast, CycleGAN is slightly more aggressive in general by causing noticeable changes to both background and foreground.

On the other hand, FID as the quantitative measurement shows that CycleGAN has a lower score of 40.99 compared to AttentionGAN with the value of 46.82. This indicates that the images generated by CycleGAN is highly correlated to the real-world checkout condition than AttentionGAN, making the CycleGAN to be more suitable for image synthesis framework in this project.

#### **4.2.2 Effect on Model Performance**

In order to evaluate the effectiveness of rendered images in detecting retail products under actual checkout condition, YOLOv5L was trained using 3 datasets that indicates 3 different levels of image synthesis used in this project. *Single* dataset involves one product placed randomly at the checkout counter whereas *Syn* dataset only simulates the occlusion between products using Crop and Place algorithm in Chapter 3.3.2. On the other hand, *Render* dataset adds the presence of shadows and lighting variation through CycleGAN as well as conventional image augmentation respectively. Each dataset was used to train YOLOv5L and Data Priming Network (FPS), which is a Faster R-CNN-based reference model by Li et al. (2019) that adopted similar image synthesis approach for their retail product recognition applications. After training the model for the same epoch number, both models were tested for scenes with and without overlapping products to justify if their performance in handling the real-world checkout condition. Their quantitative results are as tabulated in Table 4.4.

#### 4.2.2.1 Quantitative Results

Table 4.4: Quantitative Results for Different Levels of Image Synthesis

Model		Non-overlapping Scene			Overlapping Scene		
		mAP (%)	cAcc (%)	Inference Time (s)	mAP (%)	cAcc (%)	Inference Time (s)
DPNet (Baseline)	Single	98.4	56.17	1.767	65.4	0.50	1.762
	Syn	97.9	81.83	1.770	97.5	76.50	1.765
	Render	98.8	83.83	1.767	97.8	79.67	1.781
YOLOv5L	Single	99.5	94.50	0.661	89.8	39.00	0.685
	Syn	99.5	99.33	0.689	98.5	96.33	0.689
	Render	99.5	99.83	0.682	98.5	97.33	0.681

Based on Table 4.4, when the models are trained with *single* dataset, YOLOv5L outperforms DPNet in all aspects. However, it can be noticed that both models show a decrement in mAP and cAcc with the presence of overlapped products. YOLOv5L suffered a mAP drop from 99.5% to 89.8% and the cAcc decreased drastically from 94.50% to only 39.00%. while DPNet has a drop of mAP from 98.4% to 65.4% as well as reduction of cAcc from 56.17% to only 0.50%. This trend indicates that the dataset with only one product is ineffective in tackling actual checkout scenario.

Furthermore, when *Syn* dataset is used for training, YOLOv5L still surpasses DPNet but both models demonstrate a performance improvement especially in the aspect of cAcc where YOLOv5L and DPNet have the increment of 57.33% and 76% in overlapping scene respectively. At the same time, with *Syn* dataset, overall performance of YOLOv5L has been boosted to a saturation point with (mAP, cAcc) of (99.5%, 99.33%) in non-overlapping scene and (98.5%, 96.33) when overlapping products are present. This trend proves that Crop and Place algorithm can help to increase the practicality of model in detecting and counting products during occluded scene.

Moreover, when *Render* dataset is applied, YOLOv5L still has an advantage over DPNet with higher value in all aspects. However, both models

show a slight improvement compared to the scenario trained by *Syn* dataset. Concurrently, with *Render* dataset, YOLOv5L successfully achieved a peak performance where the value of mAP and cAcc are pushed to (99.5%, 99.83%) and (98.5%, 97.33%) for overlapping and non-overlapping scene respectively. The increment demonstrates that GAN-based shadow synthesis and lighting variation through conventional image augmentation can boost the capability of product recognition model in handling checkout scenario in real life.

In brief, YOLOv5L performs better compared to DPNNet in recognizing and counting retail products. It is most likely because Faster R-CNN adopts fixed, multi-scale anchor boxes while YOLOv5 family uses adaptive anchor boxes that will compute for the anchors that can best fit the current dataset, making it effective in learning and detecting variable-sized products and occlusion compared to Faster R-CNN. On the other hand, YOLOv5L also surpassed DPNNet in terms of inference time by at least 60% for all three levels of image synthesis since DPNNet is based on Faster R-CNN with a two-stage architecture. Thus, it can be inferred that GAN-based shadow synthesis and lighting variation are effective in enhancing model's practicality in real-world checkout process while YOLOv5L is more relevant to be used in in this project due to high performance with minimal inference time.

### **4.3 Model Improvement with light-weight backbones**

In order to allow deployment of YOLOv5 model on Jetson Nano with limited computational resources while preserving its recognition performance. Several experimental models were developed by replacing backbone structure of YOLOv5L with state-of-the-art light-weight CNN, including MobileNet V3, ShuffleNet V2 and GhostNet. After the training using a similar set of hyperparameters shown in Chapter 3.4.2, qualitative and quantitative evaluation were carried out and the results were tabulated in the following sections.

### 4.3.1 Quantitative Results

Table 4.5: Quantitative Results for Light-Weight Experimental Models

Backbone	GFLOPs	mAP (%)	cAcc (%)	Training	Inference Time	
				Time (hrs)	Normal (s)	TensorRT (s)
<b>CSPNet (Baseline)</b>	108.1	98.5	97.33	6.470	0.505	0.282
<b>GhostNet</b>	42.5	98.2	89.33	3.852	0.244	0.221
<b>ShuffleNet V2</b>	40.7	98.2	87.83	2.891	0.217	0.154
<b>MobileNet V3</b>	38.5	98.2	89.17	2.019	0.200	0.142

According to Table 4.5, it can be clearly observed that all light-weight backbones have successfully reduced the FLOPs of YOLOv5L baseline model to low values. In exchange, slight degradation of mAP and cAcc occurred. Among all experimental models, GhostNet with 42.5 FLOPs has the smallest degradation from the baseline performance where its mAP only falls by 0.3% while still having a high cAcc value of 89.33%. Similarly, ShuffleNet V2 with 40.7 GFLOPs has a 0.3% drop in mAP but it has the most drastic decrement of cAcc among all models, which is from 97.33% to 87.83%. Concurrently, MobileNet V3’s performance is similar to GhostNet by achieving 98.2% in mAP and 89.17% in cAcc while having a minimal FLOPs of 38.5 GLOPs among all models.

On the other hand, by looking into the training and inference time, all experimental models require shorter training time of below 4 hours, which can be advantageous compared to the baseline model that requires 6.470 hours for training. Furthermore, among the three light-weight models, MobileNet V3 is the most efficient in training since it only requires 2.019 hours while ShuffleNet V2 and GhostNet require 2.891 hours and 3.852 hours respectively. As for the normal inference time, all models are able to reduce the inference time from 0.505s to below 0.3s per image on Jetson Nano. The shortest inference time can be observed for MobileNet V3 with only 0.2s per image in contrast to

ShuffleNet V2 as well as GhostNet with 0.217s and 0.244s respectively. Furthermore, after model optimization using TensorRT runtime mentioned in Chapter 3.6.1, inference time of each model has been greatly shortened on Jetson Nano. The most significant improvement can be seen for baseline model with 44.2% improvement. Besides, other experimental models also improved by at least 29% and MobileNet V3 has achieved minimum inference time of 0.142s on Jetson Nano which is equivalent to 7 Frame Per Second (FPS).

On the other hand, performance of experimental models on every product class can be assessed through Confusion Matrix in Figure 4.2.

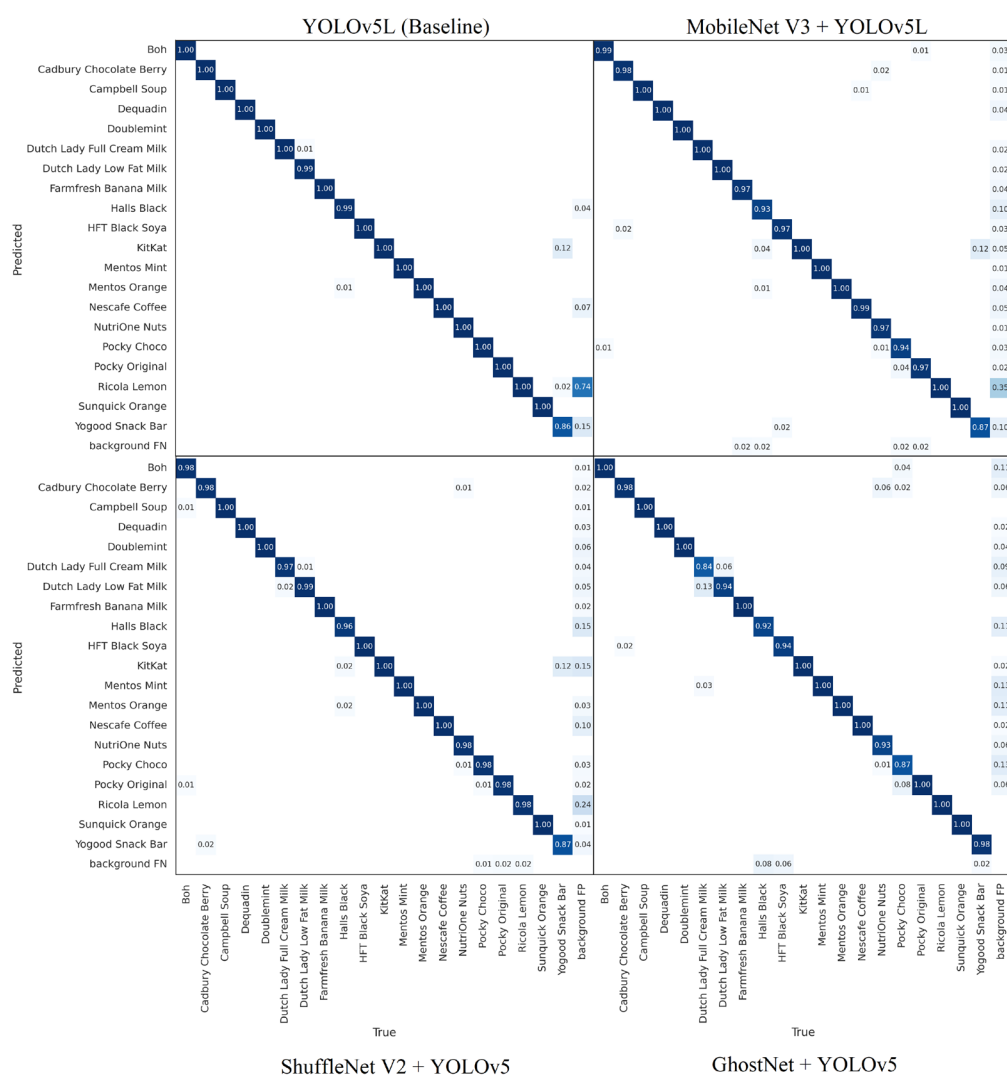


Figure 4.2: Confusion Matrix of Experimental Models

From the figure, it can be noticed that all models have more Background False Positive (FP) predictions where they detect background as

products compared to baseline model. This situation seems to be more serious for model with MobileNet V3 backbone because Background FP is present for most of the product classes compared to ShuffleNet V2 and GhostNet variant. However, these FP predictions can be avoided by increasing the confidence threshold in actual implementation. Meanwhile, ShuffleNet V2 and GhostNet are weaker in handling intraclass variation compared to MobileNet V3 because both models will have several misclassifications between *Dutch Lady Full Cream Milk* and *Dutch Lady Low Fat Milk*. Thus, MobileNet V3 is still a more relevant product recognition model compared to ShuffleNet V2 and GhostNet.

### 4.3.2 Qualitative Analysis

In this section, each light-weight model was assessed qualitatively to justify their performance in handling actual checkout condition that includes lighting variation and heavily occluded products. The analysis was done by performing inference of an image with large quantity of occluded products as well as unseen an unseen, low brightness image using each experimental model.

#### 4.3.2.1 Performance in Extreme Condition

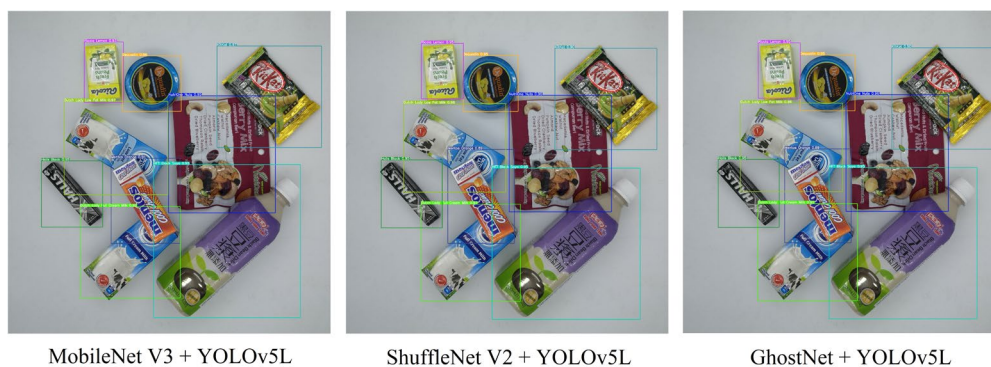


Figure 4.3: Prediction of Experimental Models in Extreme Condition

Table 4.6: Confidence Score of Detected Products in Extreme Condition

Items	Confidence Score		
	MobileNet V3	ShuffleNet V2	GhostNet
	+ YOLOv5L	+ YOLOv5L	+ YOLOv5L
<b>Dequadin</b>	0.96	0.95	0.98
<b>Dutch Lady Full Cream Milk</b>	0.92	0.92	0.96
<b>Dutch Lady Low Cream Milk</b>	0.97	0.96	0.95
<b>Halls Black</b>	0.95	0.95	0.99
<b>HFT Black Soya</b>	0.95	0.95	0.96
<b>KitKat</b>	0.97	0.90	0.98
<b>Mentos Orange</b>	0.91	0.89	0.95
<b>NutriOne Nuts</b>	0.95	0.96	0.96
<b>Ricola Lemon</b>	0.93	0.95	0.97

According to Figure 4.3, each of the experimental model successfully provides prediction for every product in extreme condition where products are heavily occluded with each other along with intraclass variation. However, it can be observed that the predicted bounding box of ShuffleNet V2 is slightly misaligned with the product compared to MobileNet V3 as well as GhostNet variant.

On the other hand, based on confidence score of each detected product shown in Table 4.5, it can be noticed that YOLOv5 with GhostNet backbone can provide predictions with high confidences since all the values are above 0.95. Besides, confidence scores provided by MobileNet V3 variant are also considered acceptable with the lowest score of 0.91. In contrast, ShuffleNet V2 gives predictions with lower confidences among all models especially for the products that are overlapped such as *KitKat* and *Mentos Orange*. Thus, it can be inferred that YOLO v5 with MobileNet V3 and GhostNet backbone are able to adapt to extreme condition of real-world checkout counter scenario.

### 4.3.2.2 Adaptivity to Lighting Variation

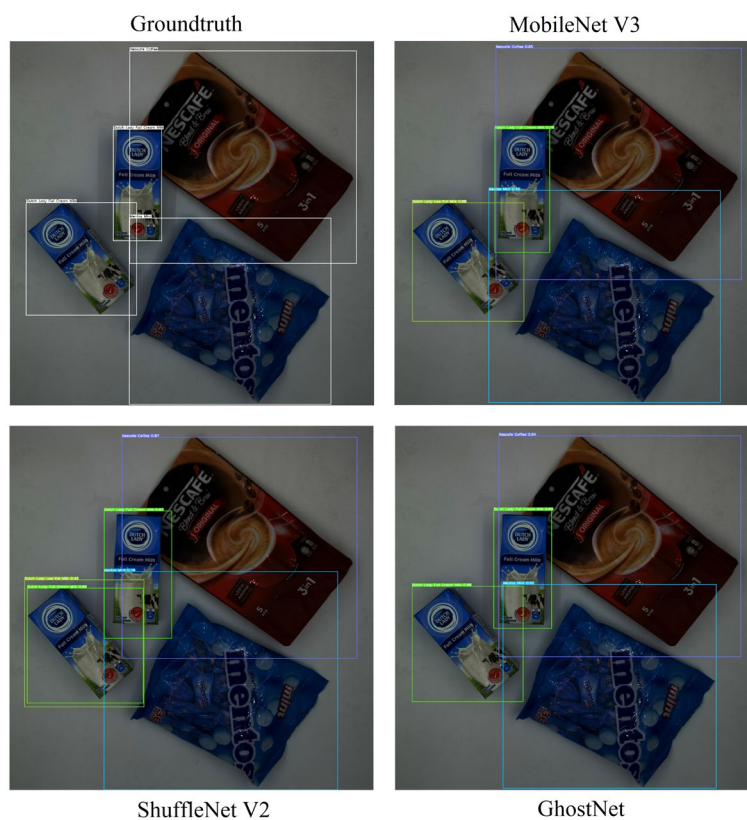


Figure 4.4: Prediction of Experimental Models in Low Light

Table 4.7: Confidence Score of Detected Products in Low Light

Items	Confidence Score		
	MobileNet V3 + YOLOv5L	ShuffleNet V2 + YOLOv5L	GhostNet + YOLOv5L
Dutch Lady Full Cream Milk	0.78	0.87	0.96
Mentos Mint	0.40	0.98	0.92
Nescafe Coffee	0.85	0.98	0.94

From Figure 4.4 which demonstrates the prediction of each experimental model in low-light condition, it is found that the bounding boxes generated by ShuffleNet V2 and GhostNet are very similar with the ground truth bounding boxes even in low-light condition. In contrast, the bounding boxes



generated by MobileNet V3 are distinct compared to ground truth boxes despite all the products are predicted accurately.

Meanwhile, according to Table 4.5 which shows the confidence score of each predicted product in the image, it is discovered that GhostNet is able to maintain its performance by providing an accurate prediction at high confidence score. In comparison, YOLOv5 model with MobileNet V3 backbone has low confidence score in general whereas ShuffleNet V2 variant has higher confidence score but the prediction is associated with a false positive of *Dutch Lady Low Fat Milk*. Therefore, it can be inferred that GhostNet variant can adapt to lighting variation effectively, followed by the model with MobileNet V3 backbone.

### 4.3.3 Training Loss

Furthermore, each experimental model can be evaluated and compared through their training and loss curve. Based on Figure 4.5 to 4.8, it can be observed that there is no overfitting occur for all models since their validation loss are able to converge and no increment happened at the end of the training process. However, in contrast to the loss curve of baseline model, all three models seem to be underfit because their training loss did not fully converge but this can be overcome by performing training for more epochs. Among all the light-weight models, training loss of GhostNet is the fastest to converge. Hence it can be inferred that GhostNet variant is more efficient in training compared to other light-weight experimental models.

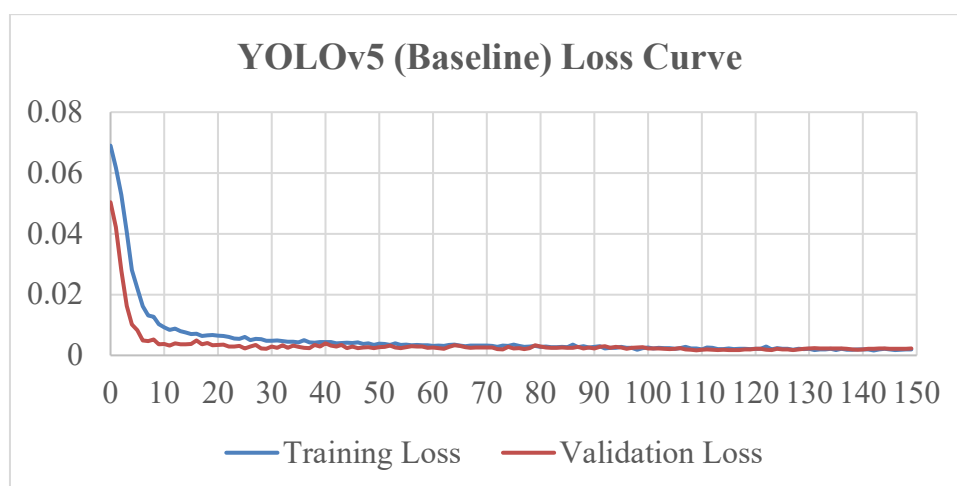


Figure 4.5: Loss Curve of YOLOv5 (Baseline)

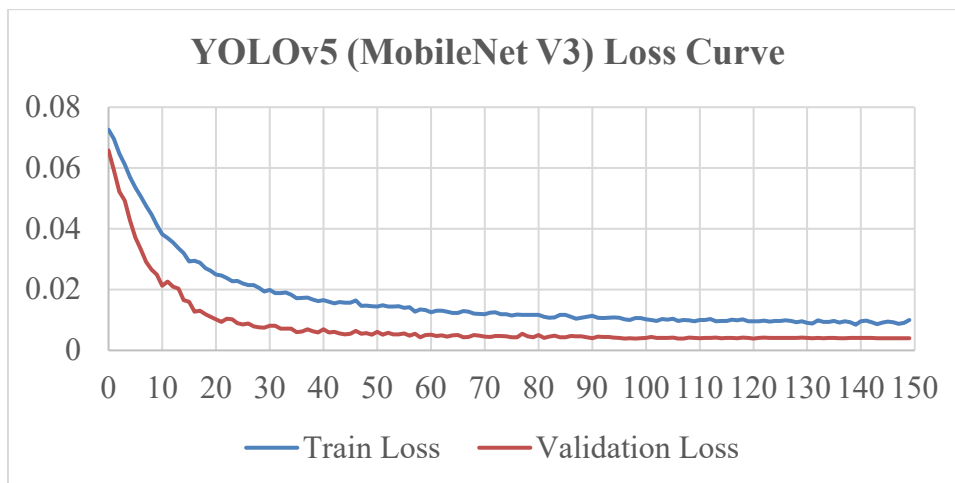


Figure 4.6: Training and Validation Loss of YOLOv5 (MobileNet V3)

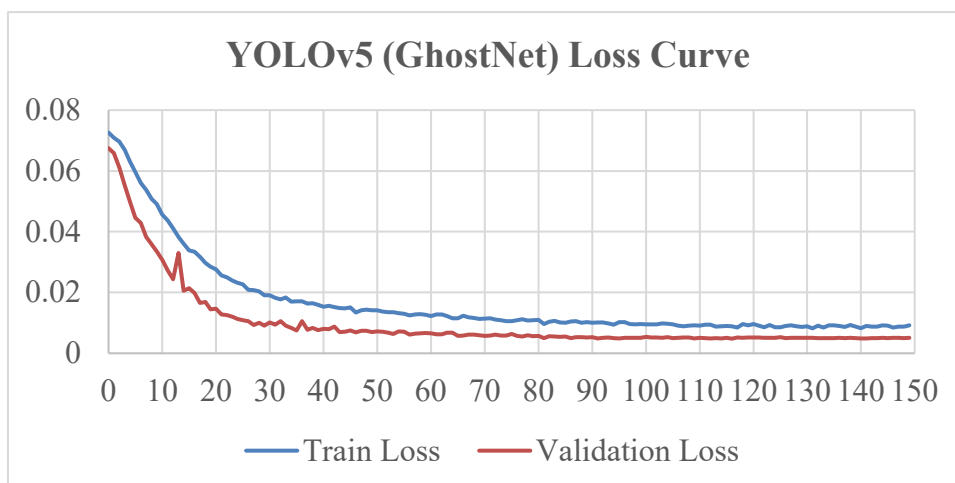


Figure 4.7: Training and Validation Loss of YOLOv5 (GhostNet)

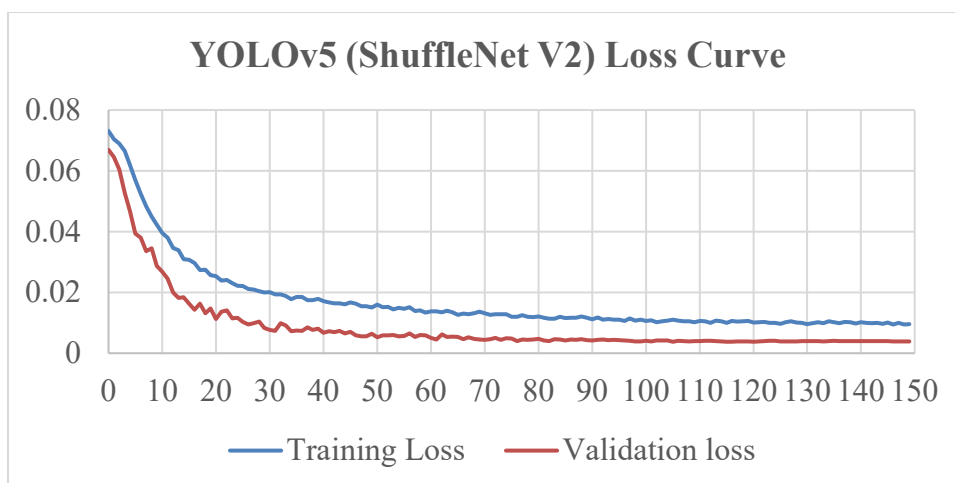


Figure 4.8: Training and Validation Loss of YOLOv5 (ShuffleNet V2)

## 4.4 Software Prototype of Cashierless Checkout System

By using the weights of baseline YOLOv5 model as well as YOLOv5 with MobileNet V3 backbone structure, a software prototype was developed in Python language, and it involves MongoDB Application Programming Interface (API) for database connection as well as Tkinter for GUI development. Each of the feature will be demonstrated in the following sections.

### 4.4.1 Model Weights Update

In order to avoid human involvement in maintaining and updating the cashierless checkout system, the proposed software prototype included a feature that will keep the product recognition model up to date before daily operation.

When a model is trained with new product category, user will just have to select the updated model weight through a python script and the file will be uploaded to MongoDB database before the file name and unique id are written into a Comma Separated Value (.csv) file. Subsequently, as shown in Figure 4.9, Jetson Nano as the edge device will fetch the latest weight to the software directory based on the file name and unique id in the Comma Separated Values (CSV) file. If there is no updated model weight in the database, the software will proceed to launch after providing a ‘No update’ output.



Figure 4.9: Model Weight Automatic Update Feature

### 4.4.2 Price Computation

As the fundamental feature of a checkout system, price computation function was added to the proposed software prototype along with other supportive elements to improve the accuracy of detection and price computation. When the

products are being placed under the camera, the software prototype compute for frame difference and contour using *createBackgroundSubtractor MOG2()* and *findContours()* shown in Figure 4.10 to detect for any object or hand movement in the area. This helps to prevent pre-mature detection and price computation that might result in false predictions and product price.



Figure 4.10: Frame Difference and Contour

Moreover, the software prototype will allow users to continue their checkout session when there are more products to be purchased. All detected products will be listed in the shopping cart section and their price is updated in real time based on the latest price fetched from MongoDB database, which is as demonstrated in Figure 4.11.

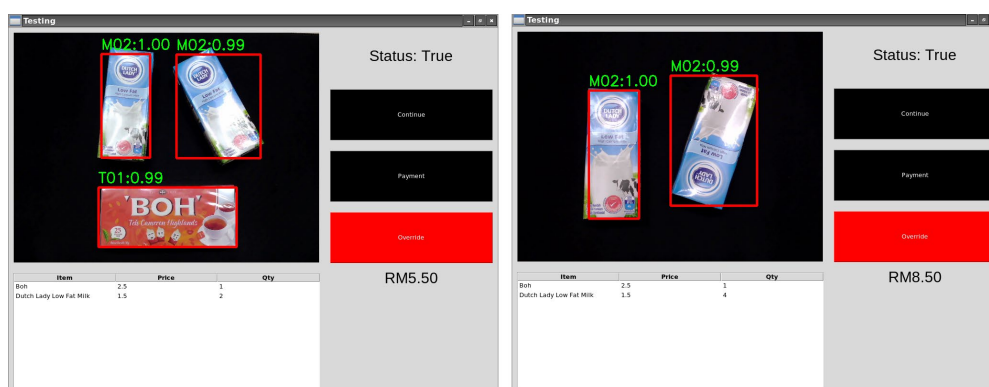


Figure 4.11: Continuous Update of Shopping Cart

## CHAPTER 5

### CONCLUSIONS AND RECOMMENDATIONS

#### 5.1 Conclusions

In a nutshell, the ultimate goal of this project is achieved where a software prototype of computer vision-based cashierless checkout system is developed and deployed on Jetson Nano by using YOLOv5 as the state-of-the-art deep learning model by fulfilling all the sub-objectives in this project.

Firstly, an image pre-processing framework is constructed successfully by involving a crop and paste algorithm to generate synthetic images with randomly placed products, followed by CycleGAN and conventional image augmentation to simulate shadows and lighting variation in images with minimal human effort. With an FID of 40.99, the framework is proved to generate training data that is closely correlated to real images. Besides, the training data also boosted the performance of YOLOv5L to 98.5% in mAP and 97.33% in cAcc in scene with overlapping products.

Subsequently, a deep learning-based product recognition model is successfully built by involving YOLOv5L as its achieved uniformly high mAP of 99.5% while maintaining short inference time of 12.20ms and training time of 5.69 hours in the preliminary benchmark with other single-stage representative models on MVTec D2S Dataset.

Moreover, YOLOv5L was further improved with several light-weight backbone architecture, including MobileNet V3, ShuffleNet V2 and GhostNet. The result demonstrates that the inference time was effectively shortened to 0.142s through TensorRT runtime with only degradation of 0.3% in mAP and cAcc is maintained at 89.17% with the lightest MobileNet V3 architecture.

Lastly, the product recognition model is successfully implemented in a software prototype using Tkinter along with several features like price retrieval and computation function as well as model weight update capability through MongoDB database API. The results demonstrates that the software prototype can run successfully even on edge devices like Jetson Nano and able to provide accurate predictions and price computation.

## 5.2 Recommendations for Future Work

Due to time constraint, the results presented in this project may not be optimal. There are multiple works that can be carried out to elevate the quality of this project. First of all, the existing image synthesis framework can be further enhanced by using adopting other GAN architectures or techniques that can simulate more realistic shadow and lighting variation while having less background and product color degradation.

Moreover, the light-weight models used for product recognition can be further improved with state-of-the-art modules or functions to obtain a better trade-off in terms of accuracy and inference time on edge devices. For instance, attention module or model pruning can be applied to improve the accuracy while maintaining the model size and FLOPs.

Additionally, incremental learning can be applied in the training of product recognition model as an effort in tackling the actual retail store challenges where new products or package design are introduced continuously.

Lastly, the software prototype of cashierless checkout system can be enhanced by including more features like e-payment gateway and auto-packaging system to further reduce the work force requirement at the checkout counter.

## REFERENCES

Amazon, 2016. *Introducing Amazon Go and the world's most advanced shopping technology*, [online] Available at: <<https://www.youtube.com/watch?v=NrmMk1Myrxc>> [Accessed 16 August 2021].

Amsler, S. and Shea, S., 2021. *What is RFID and how does it work?*, [online] IoT Agenda. Available at: <<https://internetofthingsagenda.techtarget.com/definition/RFID-radio-frequency-identification>> [Accessed 10 August 2021].

Ananth, S., 2020. *Faster R-CNN for object detection*. Medium, [online] Available at: <<https://towardsdatascience.com/faster-r-cnn-for-object-detection-a-technical-summary-474c5b857b46>> [Accessed 28 August 2021].

Anwla, P.K., 2020. *Focal Loss in Object Detection | A Guide To Focal Loss*. Analytics Vidhya, [online] Available at: <<https://www.analyticsvidhya.com/blog/2020/08/a-beginners-guide-to-focal-loss-in-object-detection/>> [Accessed 30 August 2021].

Bocanegra, C., Khojastepour, M.A. (Amir), Arslan, M.Y., Chai, E., Rangarajan, S. and Chowdhury, K.R., 2020. RFGo: a seamless self-checkout system for apparel stores using RFID. In: *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. [online] pp.1–14. <https://doi.org/10.1145/3372224.3419211>.

Bochkovskiy, A., Wang, C.-Y. and Liao, H.-Y.M., 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv:2004.10934 [cs, eess]*, [online] Available at: <<http://arxiv.org/abs/2004.10934>> [Accessed 2 September 2021].

Boesch, D., 2021. *Object Detection in 2021: The Definitive Guide*. [online] Available at: <<https://viso.ai/deep-learning/object-detection/>> [Accessed 1 September 2021].

Bukhari, S.T., Amin, A.W., Naveed, M.A. and Abbas, M.R., 2021. ARC: A Vision-based Automatic Retail Checkout System. *arXiv:2104.02832 [cs]*, [online] Available at: <<http://arxiv.org/abs/2104.02832>> [Accessed 11 August 2021].

Buslaev, A., Iglovikov, V.I., Khvedchenya, E., Parinov, A., Druzhinin, M. and Kalinin, A.A., 2020. Albumentations: Fast and Flexible Image Augmentations. *Information*, 11(2), p.125. <https://doi.org/10.3390/info11020125>.

Chauhan, N.K. and Singh, K., 2018. A Review on Conventional Machine Learning vs Deep Learning. In: *2018 International Conference on Computing, Power and Communication Technologies (GUCON)*, [online] IEEE.pp.347–352. <https://doi.org/10.1109/GUCON.2018.8675097>.

Davis, B., 2021. *What is the state of the art in research? – Mvorganizing.org*. [online] Available at: <[https://www.mvorganizing.org/what-is-the-state-of-the-art-in-research2/#What\\_is\\_the\\_state\\_of\\_the\\_art\\_in\\_research](https://www.mvorganizing.org/what-is-the-state-of-the-art-in-research2/#What_is_the_state_of_the_art_in_research)> [Accessed 1 September 2021].

Deshmukh, A., 2020. *RetinaNet Model for object detection explanation*. [online] Available at: <<https://towardsmachinelearning.org/retinanet-model-for-object-detection-explanation/>> [Accessed 3 September 2021].

Dodge, S. and Karam, L., 2016. Understanding how image quality affects deep neural networks. In: *2016 Eighth International Conference on Quality of Multimedia Experience (QoMEX)*. [online] 2016 Eighth International Conference on Quality of Multimedia Experience (QoMEX). Lisbon, Portugal: IEEE.pp.1–6. <https://doi.org/10.1109/QoMEX.2016.7498955>.

Dubovikov, K., 2018. *PyTorch vs TensorFlow — spotting the difference*. [online] Medium. Available at: <<https://towardsdatascience.com/pytorch-vs-tensorflow-spotting-the-difference-25c75777377b>> [Accessed 17 April 2022].

Dutta, A. and Zisserman, A., 2019. The VIA Annotation Software for Images, Audio and Video. In: *Proceedings of the 27th ACM International Conference on Multimedia, MM '19*. [online] New York, NY, USA: Association for Computing Machinery.pp.2276–2279.<https://doi.org/10.1145/3343031.3350535>.

Everingham, M. and Winn, J., 2012. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Development Kit. *Pattern Analysis, Statistical Modelling and Computational Learning, Tech. Rep*, 8, p.32.



Farfan, B., 2020. *What Is Retail?* [online] Available at: <<https://www.thebalancesmb.com/what-is-retail-2892238>> [Accessed 5 September 2021].

Follmann, P., Böttger, T., Härtinger, P., König, R. and Ulrich, M., 2018. *MVTec D2S: Densely Segmented Supermarket Dataset*. In: Proceedings of the European Conference on Computer Vision (ECCV). [online] pp.581–585. Available at: <[http://link.springer.com/10.1007/978-3-030-01249-6\\_35](http://link.springer.com/10.1007/978-3-030-01249-6_35)> [Accessed 29 July 2021].

Geng, W., Han, F., Lin, J., Zhu, L., Bai, J., Wang, S., He, L., Xiao, Q. and Lai, Z., 2018. *Fine-Grained Grocery Product Recognition by One-Shot Learning*. In: Proceedings of the 26th ACM international conference on Multimedia. [online] pp.1706–1714. <https://doi.org/10.1145/3240508.3240522>.

Gollapudi, S., 2019. *Artificial Intelligence and Computer Vision*. In: Learn Computer Vision Using OpenCV. [online] Berkeley, CA: Apress.pp.1–29. [https://doi.org/10.1007/978-1-4842-4261-2\\_1](https://doi.org/10.1007/978-1-4842-4261-2_1).

Google, n.d. *Frequently Asked Questions*. [online] Available at: <<https://research.google.com/colaboratory/faq.html>> [Accessed 14 November 2021].

Han, K., Wang, Y., Tian, Q., Guo, J., Xu, C. and Xu, C., 2020. GhostNet: More Features from Cheap Operations. *arXiv:1911.11907 [cs]*, [online] Available at: <<http://arxiv.org/abs/1911.11907>> [Accessed 17 April 2022].

Hashemi, M., 2019. Enlarging smaller images before inputting into convolutional neural network: zero-padding vs. interpolation. *Journal of Big Data*, 6(1), p.98. <https://doi.org/10.1186/s40537-019-0263-7>.

He, K., Gkioxari, G., Dollár, P. and Girshick, R., 2018. Mask R-CNN. *arXiv:1703.06870 [cs]*, [online] Available at: <<http://arxiv.org/abs/1703.06870>> [Accessed 29 August 2021].

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B. and Hochreiter, S., 2018. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. *arXiv:1706.08500 [cs, stat]*, [online] Available at: <<http://arxiv.org/abs/1706.08500>> [Accessed 17 April 2022].

Hirschmann, R., 2021. *Malaysia: COVID-19 MCO movements 2020*. [online] Available at: <<https://www-statista-com.libdb.njit.edu:8443/statistics/1117078/malaysia-movement-and-behavior-during-mco-covid-19/>> [Accessed 26 August 2021].

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q.V. and Adam, H., 2019. Searching for MobileNetV3. *arXiv:1905.02244 [cs]*. [online] Available at: <<http://arxiv.org/abs/1905.02244>> [Accessed 6 April 2022].

IBM Cloud Education, 2020. *What is Machine Learning?* [online] IBM. Available at: <<https://www.ibm.com/cloud/learn/machine-learning>> [Accessed 24 August 2021].

International Business Machine Corporation (IBM), n.d. *What is Computer Vision?* [online] Available at: <<https://www.ibm.com/topics/computer-vision>> [Accessed 16 August 2021].

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

Jund, P., Abdo, N., Eitel, A. and Burgard, W., 2016. The Freiburg Groceries Dataset. *arXiv:1611.05799 [cs]*, [online] Available at: <<http://arxiv.org/abs/1611.05799>> [Accessed 29 July 2021].

Khan, S., Rahmani, H., Shah, S.A.A. and Bennamoun, M., 2018. A Guide to Convolutional Neural Networks for Computer Vision. *Synthesis Lectures on Computer Vision*, 8(1), pp.1–207. <https://doi.org/10.2200/S00822ED1V01Y201712COV015>.

Koturwar, S., Shiraishi, S. and Iwamoto, K., 2019. Robust multi-object detection based on data augmentation with realistic image synthesis for point-of-sale automation. *33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019*, pp.9492–9497. <https://doi.org/10.1609/aaai.v33i01.33019492>.

Li, C., Du, D., Zhang, L., Luo, T., Wu, Y., Tian, Q., Wen, L. and Lyu, S., 2019. Data Priming Network for Automatic Check-Out. *arXiv:1904.04978 [cs]*, [online] Available at: <<http://arxiv.org/abs/1904.04978>> [Accessed 7 August 2021].

Lin, T.Y., Goyal, P., Girshick, R., He, K. and Dollár, P., 2018. Focal Loss for Dense Object Detection. *arXiv:1708.02002 [cs]*, pp.1–29.

Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X. and Pietikäinen, M., 2020. Deep Learning for Generic Object Detection: A Survey. *International Journal of Computer Vision*, 128(2), pp.261–318. <https://doi.org/10.1007/s11263-019-01247-4>.

Liu, W., Guo, J., Lin, H., Huang, L. and Gao, Z., 2019. A Bread Recognition System Based on Faster R-CNN. *Journal of Computers*, 30(6), pp.216–222.  
Lutz, M., 2006. *Programming Python*. 3rd ed ed. [online] O'Reilly. Available at: <[libgen.li/file.php?md5=9487d2bd841af34bd9034639df930683](http://libgen.li/file.php?md5=9487d2bd841af34bd9034639df930683)> [Accessed 17 April 2022].

Ma, N., Zhang, X., Zheng, H.-T. and Sun, J., 2018. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. *arXiv:1807.11164 [cs]*, [online] Available at: <<http://arxiv.org/abs/1807.11164>> [Accessed 14 April 2022].

Mantripragada, M., 2020. *Digging deep into YOLO V3 — A hands-on guide Part 1*. [online] Available at: <<https://towardsdatascience.com/digging-deep-into-yolo-v3-a-hands-on-guide-part-1-78681f2c7e29>> [Accessed 3 September 2021].

Ministry of Health Malaysia, 2020. *COVID-19: Management Guidelines For Workplaces*. Available at: <[https://www.moh.gov.my/moh/resources/Penerbitan/Garis%20Panduan/COVID19/Annex\\_25\\_COVID\\_guide\\_for\\_workplaces\\_22032020.pdf](https://www.moh.gov.my/moh/resources/Penerbitan/Garis%20Panduan/COVID19/Annex_25_COVID_guide_for_workplaces_22032020.pdf)> [Accessed 13 November 2021].

Moorthy, R., Behera, S., Verma, S., Bhargave, S. and Ramanathan, P., 2015. Applying Image Processing for Detecting On-Shelf Availability and Product Positioning in Retail Stores. In: *Proceedings of the Third International Symposium on Women in Computing and Informatics*. pp.451–457. <https://doi.org/10.1145/2791405.2791533>.

Ning, J., Li, Y. and Ramesh, A., 2019. Simplifying Grocery Checkout with Deep Learning. [online] Available at: <<https://www.semanticscholar.org/paper/Simplifying-Grocery-Checkout-with-Deep-Learning-Ning-Li/975ff328a7c86b593ed2501a1dd33a43e629b8bc>> [Accessed 29 August 2021].

NVIDIA, 2014. Jetson Nano Data Sheet. [online] Available at: <[https://developer.download.nvidia.com/assets/embedded/secure/jetson/Nano/docs/JetsonNano\\_DataSheet\\_DS09366001v1.0.pdf](https://developer.download.nvidia.com/assets/embedded/secure/jetson/Nano/docs/JetsonNano_DataSheet_DS09366001v1.0.pdf)> [Accessed 6 April 2022]

NVIDIA, 2016. *NVIDIA TensorRT*. [online] Available at: <<https://developer.nvidia.com/tensorrt>> [Accessed 6 April 2022].

Oh, J.S. and Chun, I.G., 2020. Implementation of Smart Shopping Cart using Object Detection Method based on Deep Learning. *Journal of the Korea Academia-Industrial cooperation Society*, 21(7), pp.262–269. <https://doi.org/10.5762/KAIS.2020.21.7.262>.

Ohri, A., 2021. *YOLO Algorithm For Object Detection: A Simple Guide (2021)*. [online] Jigsaw Academy. Available at: <<https://www.jigsawacademy.com/blogs/ai-ml/yolo-algorithm/>> [Accessed 29 August 2021].

O'Mahony, N., Campbell, S., Carvalho, A., Harapanahalli, S., Hernandez, G.V., Krpalkova, L., Riordan, D. and Walsh, J., 2019. Deep Learning vs. Traditional Computer Vision. In: *Science and Information Conference*. pp.128–144. [https://doi.org/10.1007/978-3-030-17795-9\\_10](https://doi.org/10.1007/978-3-030-17795-9_10).

Panasonic, 2018. *RFID Based Walk-through Checkout Solution for Future Retail*. [online] Panasonic Newsroom Global. Available at: <<http://news.panasonic.com/global/topics/2018/55288.html>> [Accessed 14 August 2021].

Potter, R., 2021. *What is the use and purpose of image annotation in object detection?* [online] Medium. Available at: <<https://becominghuman.ai/what-is-the-use-and-purpose-of-image-annotation-in-object-detection-8b7873a14cd0>> [Accessed 16 April 2022].

Raspberry Pi, 2019. Raspberry Pi 4 Datasheet. [online] Available at: <<https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf>> [Accessed 17 April 2022].

Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You Only Look Once: Unified, Real-Time Object Detection. *arXiv:1506.02640 [cs]*, [online] Available at: <<http://arxiv.org/abs/1506.02640>> [Accessed 29 August 2021].

Redmon, J. and Farhadi, A., 2018. YOLOv3: An Incremental Improvement. *arXiv:1804.02767 [cs]*, [online] Available at: <<http://arxiv.org/abs/1804.02767>> [Accessed 3 September 2021].

Ren, S., He, K., Girshick, R. and Sun, J., 2016. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *arXiv:1506.01497 [cs]*, [online] Available at: <<http://arxiv.org/abs/1506.01497>> [Accessed 11 August 2021].

Rigner, A., 2019. AI-based machine vision for retail self-checkout system. *Master's Theses in Mathematical Sciences*. [online] Available at: <<http://lup.lub.lu.se/student-papers/record/8985308>> [Accessed 4 August 2021].

Sammut, C. and Webb, G.I. eds., 2017. *Encyclopedia of Machine Learning and Data Mining*. [online] Boston, MA: Springer US. <https://doi.org/10.1007/978-1-4899-7687-1>.

Saxena, P., 2020. *Data Augmentation for Custom Object Detection | YOLO*. [online] Available at: <<https://medium.com/predict/data-augmentation-for-custom-object-detection-15674966e0c8>> [Accessed 4 August 2021].

Seals, M., 2019. *On the Robustness of Object Detection Based Deep Learning Models*. [online] University of Tennessee. Available at: <[https://trace.tennessee.edu/utk\\_gradthes/5487](https://trace.tennessee.edu/utk_gradthes/5487)> [Accessed 21 August 2021].

Sherman, R., 2014. *Business Intelligence Guidebook*. 1st ed. [online] Morgan Kaufmann. Available at: <<https://www.sciencedirect.com/science/article/pii/B9780124114616000186>>.

Sweeney, P.J., 2010. *RFID For Dummies*. [online] Indiana: Wiley Publishing, Inc. Available at: <<https://www.wiley.com/en-us/RFID+For+Dummies-p-9781118054475>> [Accessed 10 August 2021].

Tang, H., Liu, H., Xu, D., Torr, P.H.S. and Sebe, N., 2021. AttentionGAN: Unpaired Image-to-Image Translation using Attention-Guided Generative Adversarial Networks. *arXiv:1911.11897 [cs, eess]*, [online] Available at: <<http://arxiv.org/abs/1911.11897>> [Accessed 16 April 2022].

Thuan, D., 2021. Evolution of yolo algorithm and yolov5: the state-of-the-art object detection algorithm. [online] Available at: <<http://urn.fi/URN:NBN:fi:amk-202103042892>>.

Tzutalin, 2015. *LabelImg*. [Python] Available at: <<https://github.com/tzutalin/labelImg>> [Accessed 5 September 2021].

Varol, G. and Salih, R., 2015. Toward retail product recognition on grocery shelves. p.944309. <https://doi.org/10.1117/12.2179127>.

Wang, C.Y., Liao, H.Y.M., Yeh, I.H., Wu, Y.H., Chen, P.Y. and Hsieh, J.-W., 2019. CSPNet: A New Backbone that can Enhance Learning Capability of CNN. *arXiv:1911.11929 [cs]*, [online] Available at: <<http://arxiv.org/abs/1911.11929>> [Accessed 3 September 2021].

Wei, X.S., Cui, Q., Yang, L., Wang, P. and Liu, L., 2019. RPC: A Large-Scale Retail Product Checkout Dataset. *arXiv:1901.07249 [cs]*, [online] Available at: <<http://arxiv.org/abs/1901.07249>> [Accessed 28 July 2021].

Wei, Y., Tran, S., Xu, S., Kang, B. and Springer, M., 2020. Deep Learning for Retail Product Recognition: Challenges and Techniques. *Computational Intelligence and Neuroscience*, 2020, pp.1–23. <https://doi.org/10.1155/2020/8875910>.

Wu, B.F., Tseng, W.J., Chen, Y.S., Yao, S.J. and Chang, P.J., 2016. An intelligent self-checkout system for smart retail. In: *2016 International Conference on System Science and Engineering (ICSSE)*. IEEE. pp.1–4. <https://doi.org/10.1109/ICSSE.2016.7551621>.

Xie, L., Wang, S. and Zhao, L., 2021. Analysis of Commodity image recognition based on deep learning. In: *2021 6th International Conference on Multimedia and Image Processing*. [online] New York, NY, USA: Association for Computing Machinery. pp.50–55. Available at: <<https://doi.org/10.1145/3449388.3449389>> [Accessed 28 August 2021].

Xu, R., Lin, H., Lu, K., Cao, L. and Liu, Y., 2021. A Forest Fire Detection System Based on Ensemble Learning. *Forests*, 12, p.217. <https://doi.org/10.3390/f12020217>.

Yamashita, R., Nishio, M., Do, R.K.G. and Togashi, K., 2018. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9(4), pp.611–629. <https://doi.org/10.1007/s13244-018-0639-9>.

Yi, W., Sun, Y., Ding, T. and He, S., 2019. Detecting retail products in situ using CNN without human effort labeling. *arXiv:1904.09781 [cs]*, [online] Available at: <<http://arxiv.org/abs/1904.09781>> [Accessed 4 August 2021].

You, K., Long, M., Wang, J. and Jordan, M.I., 2019. How Does Learning Rate Decay Help Modern Neural Networks? *arXiv:1908.01878 [cs, stat]*, [online] Available at: <<http://arxiv.org/abs/1908.01878>> [Accessed 1 September 2021].

Zaidi, S.S.A., Ansari, M.S., Aslam, A., Kanwal, N., Asghar, M. and Lee, B., 2021. A Survey of Modern Deep Learning based Object Detection Models. *arXiv preprint arXiv:2104.11892*, [online] Available at: <<https://arxiv.org/abs/2104.11892v2>> [Accessed 1 September 2021].

Zhong, S., 2021. *Automatic Retail Product Identification System for Cashierless Stores*. [online] Available at: <<http://urn.kb.se/resolve?urn=urn:nbn:se:hj:diva-52659>> [Accessed 14 August 2021].

Zhu, J.Y., Park, T., Isola, P. and Efros, A.A., 2020. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *arXiv:1703.10593 [cs]*, [online] Available at: <<http://arxiv.org/abs/1703.10593>> [Accessed 7 August 2021].

Zou, Z., Shi, Z., Guo, Y. and Ye, J., 2019. Object Detection in 20 Years: A Survey. *arXiv preprint arXiv:1905.05055*, [online] Available at: <<https://arxiv.org/abs/1905.05055v2>> [Accessed 1 September 2021].



## APPENDICES

### Appendix A: Image Synthesis Script

```

# Mount Onedrive
!curl https://rclone.org/install.sh | sudo bash
!rclone config
!sudo mkdir /content/OneDrive
!nohup rclone --vfs-cache-mode writes mount onedrive: /content/OneDrive &

# Upload Raw Images
import os

dir_list = ['images', 'mask', 'txt']

for dir in dir_list:
    path = f'/content/data/{dir}'
    if not os.path.exists(path):
        os.makedirs(path)
    else:
        print(f'"{path}" exists')

# Extract Mask using COCO annotation
%cd /content
!python Mask_Extraction_COCO.py

# Image Synthesis Script
!python synthesize_images_COCO.py --count 3000

# Move synthesized images to new directory
%cd /content
!mkdir synthesized_dataset
%cd /content/synthesized_dataset
!mkdir annotations
!cp -av "/content/synthesized_images" "/content/synthesized_dataset/images"
!cp "/content/sythesized.json" "/content/synthesized_dataset/annotations/annotation.json"

# Export synthesized images
from google.colab import files

!zip -r /content/synthesized_dataset.zip /content/synthesized_dataset
files.download("/content/synthesized_dataset.zip")

```

```

import glob
import json
import os
import random
from numpy.lib.shape_base import column_stack
import scipy
from scipy import spatial
import time
from argparse import ArgumentParser
from collections import defaultdict
import cv2
import numpy as np
from PIL import Image
from scipy import ndimage
from tqdm import tqdm
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow

```

```

NUM_CATEGORIES = 20
GENERATED_NUM = 3000

def buy_strategic(counter):
    categories = [i + 1 for i in range(NUM_CATEGORIES)]
    selected_categories = np.random.choice(categories, size=random.randint(3, 3),
                                          replace=False)
    num_categories = len(selected_categories)

    if 3 ≤ num_categories < 5:
        num_instances = 3
        counter['To be synthesized'] += 1

    num_per_category = {}
    generated = 0

    for i, category in enumerate(selected_categories):
        i += 1
        if i == num_categories:
            count = num_instances - generated
        else:
            count = random.randint(1, num_instances - (num_categories - i) - generated)
        generated += count
        num_per_category[int(category)] = count

    return num_per_category

def check_iou(annotations, box, threshold=0.5):
    cx1, cy1, cw, ch = box
    cx2, cy2 = cx1 + cw, cy1 + ch
    carea = cw * ch
    for ann in annotations:
        x1, y1, w, h = ann['bbox']
        x2, y2 = x1 + w, y1 + h
        area = w * h
        inter_x1 = max(x1, cx1)
        inter_y1 = max(y1, cy1)
        inter_x2 = min(x2, cx2)
        inter_y2 = min(y2, cy2)

        inter_area = max(0, inter_x2 - inter_x1) * max(0, inter_y2 - inter_y1)
        iou = inter_area / (carea + area - inter_area + 1e-8) # avoid division by zero
        if iou > threshold:
            return False
    return True

def sample_select_object_index(category, paths, ratio_annotations, threshold=0.45):
    high_threshold_paths = [path for path in paths if
                            ratio_annotations[os.path.basename(path)] > threshold]
    index = random.randint(0, len(high_threshold_paths) - 1)
    path = high_threshold_paths[index]
    return path

def get_category(path):
    class_list = read_txt(path)
    categories=[]
    for i, category in enumerate(class_list, 1):
        categories.append({'supercategory': category, 'id': i, 'name': category})
    return categories

```

```

def synthesize(strategics, save_json_file='', output_dir='', save_mask=False):
    with open('/content/dataset/annotations/instances_train2017.json') as fid:
        data = json.load(fid)
        images = {}
        for x in data['images']:
            images[x['id']] = x

        annotations = {}
        for x in data['annotations']:
            annotations[images[x['image_id']]['file_name']] = x

    object_paths = glob.glob(os.path.join('/content/dataset/train2017/', '*.jpg'))

    object_category_paths = defaultdict(list)
    for path in object_paths:
        name = os.path.basename(path)
        category = annotations[name]['category_id']
        object_category_paths[category].append(path)
    object_category_paths = dict(object_category_paths)

    bg_img_cv = cv2.imread('bg.jpg')
    bg_height, bg_width = bg_img_cv.shape[:2]
    mask_img_cv = np.zeros((bg_height, bg_width), dtype=np.uint8)

    json_ann = []
    images = []
    categories=[]

    for image_id, num_per_category in tqdm(strategics):
        bg_img = Image.fromarray(bg_img_cv)
        mask_img = Image.fromarray(mask_img_cv)
        synthesize_annotations = []
        for category, count in num_per_category.items():
            category = int(category)
            for _ in range(count):
                paths = object_category_paths[category]
                object_path = sample_select_object_index(category, paths,
                                                            ratio_annotations,
                                                            threshold=0.45)

                name = os.path.basename(object_path)
                mask_path = os.path.join('/content/dataset/train2017/',
                                         '{}.png'.format(name.split('.')[0]))

                obj = Image.open(object_path)
                mask = Image.open(mask_path).convert('L')

                angle = random.random() * 360
                obj = obj.rotate(angle, resample=Image.BILINEAR, expand=1)
                mask = mask.rotate(angle, resample=Image.BILINEAR, expand=1)

                where = np.where(np.array(mask))

                y1, x1 = np.amin(where, axis=1)
                y2, x2 = np.amax(where, axis=1)
                obj = obj.crop((x1, y1, x2, y2))
                mask = mask.crop((x1, y1, x2, y2))
                w, h = obj.width, obj.height

                pad = 0
                pos_x, pos_y = generated_position(bg_width, bg_height, w, h, pad)
                start = time.time()
                threshold = 0.1

```

```

while not check_iou(synthesize_annotations, box=(pos_x, pos_y, w, h),
                    threshold=threshold):
    if (time.time() - start) > 3:
        start = time.time()
        threshold += 0.1
        continue
    pos_x, pos_y = generated_position(bg_width, bg_height, w, h, pad)

    bg_img.paste(obj, box=(pos_x, pos_y), mask=mask)
    if save_mask:
        mask_img.paste(mask, box=(pos_x, pos_y), mask=mask)

    mask_array = np.array(mask)
    center_of_mass = ndimage.measurements.center_of_mass(mask_array)
    center_of_mass = [int(round(x)) for x in center_of_mass]
    center_of_mass = center_of_mass[1] + pos_x, center_of_mass[0] + pos_y

    synthesize_annotations.append({ 'image_id': int(image_id.split('_')[2]),
                                    'bbox': (pos_x, pos_y, w, h),
                                    'category_id': category,
                                    'center_of_mass': center_of_mass})

assert bg_height == 3200 and bg_width == 3200
scale = 200.0 / 3200
gt = np.zeros((round(bg_height * scale), round(bg_width * scale)))
for item in synthesize_annotations:
    center_of_mass = item['center_of_mass']
    gt[round(center_of_mass[1] * scale), round(center_of_mass[0] * scale)] = 1

assert gt.shape[0] == 200 and gt.shape[1] == 200

image_name = '{}.jpg'.format(image_id)
images.append({'file_name': image_name,
               'id': int(image_id.split('_')[2]),
               'height': bg_height,
               'width': bg_width})

bg_img.save(os.path.join(output_dir, image_name))

if save_mask:
    mask_img.save(os.path.join(output_dir, 'masks', image_name))
json_ann.extend(synthesize_annotations)
categories = get_category('/content/dataset/classes.txt')
json_data = {'images': images,
             'annotations': json_ann,
             'categories': categories}
if save_json_file:
    with open(save_json_file, 'w') as fid:
        json.dump(json_data, fid)

if __name__ == '__main__':
    parser = ArgumentParser(description="Synthesize fake images")
    parser.add_argument('--count', type=int, default=32)
    parser.add_argument('--local_rank', type=int, default=0)
    args = parser.parse_args()

    counter = {'To be synthesized': 0}
    strategics = []
    for image_id in tqdm(range(GENERATED_NUM)):
        num_per_category = buy_strategic(counter)
        strategics.append(('synthesized_image_{}'.format(image_id), num_per_category))

```

```
if os.path.exists('strategics.json'):
    os.remove('strategics.json')
with open('strategics.json', 'w') as f:
    json.dump(strategics, f)

with open('strategics.json') as f:
    strategics = json.load(f)
strategics = sorted(strategics, key=lambda s: s[0])

output_dir = os.path.join('synthesize_images')
if not os.path.exists(output_dir):
    os.mkdir(output_dir)

threads = []
num_threads = args.count
sub_strategics = strategics[args.local_rank::num_threads]
save_file = 'synthesize_images.json'
synthesize(sub_strategics, save_file, output_dir)
```

## Appendix B: Image Augmentation Script

```

import albumentations as A
from imgaug.augmentables.bbs import BoundingBox, BoundingBoxesOnImage
import cv2
import os
import imgaug as ia
import imgaug.augmenters as iaa
import math
import random
import copy
import glob
import argparse

class Augment:
    def __init__(self, images_dir, anno_dir, mask_dir, n, choice):
        self.images_path = images_dir
        self.anno_path = anno_dir
        self.mask_path = mask_dir
        self.N = int(n)
        self.imgtype = choice

        imgs_save_path = '/content/augmented/images'
        if not os.path.exists(imgs_save_path):
            os.makedirs(imgs_save_path)
        txt_save_path = '/content/augmented/txt'
        if not os.path.exists(txt_save_path):
            os.makedirs(txt_save_path)
        mask_save_path = '/content/augmented/masks'
        if not os.path.exists(mask_save_path):
            os.makedirs(mask_save_path)

        random.seed(7)

    def readImage(self, filename):
        print(filename)
        img = cv2.imread(filename)
        return img

    def readYolo(self, filename):
        coords = []
        with open(filename, 'r') as fname:
            for file1 in fname:
                x = file1.strip().split(' ')
                x.append(x[0])
                x.pop(0)
                x[0] = float(x[0])
                x[1] = float(x[1])
                x[2] = float(x[2])
                x[3] = float(x[3])
                coords.append(x)
        return coords

    def getTransform(self):
        try:
            if self.imgtype == "syn":
                transform = A.Compose([A.RandomRotate90(p=1), A.Resize(height=2300,
                    width=2300, interpolation=cv2.INTER_AREA),
                    A.ShiftScaleRotate(scale_limit=[0, 0], rotate_limit=45,
                    border_mode=cv2.BORDER_CONSTANT,
                    mask_value=0, shift_limit=0),
                    A.VerticalFlip(p=0),
                    A.MotionBlur(blur_limit=5, p=0.8),
                    A.RandomToneCurve(scale=0.5, p=1),
                    A.ISONoise(color_shift=(0.01, 0.05), intensity=(0.1,
                    0.5), p=0.0),

```

```

        A.RandomBrightnessContrast(brightness_limit=0.0,
                                   contrast_limit=0.0,
                                   p=0.5), ],
        bbox_params=A.BboxParams(format='yolo'))
    if self.imgtype == "real":
        transform = A.Compose([A.RandomRotate90(p=0.9), A.Resize(height=3200,
                                                                width=3200, interpolation=cv2.INTER_AREA),
                              A.ShiftScaleRotate(scale_limit=[0, 0],
                                                  rotate_limit=3, p=1,
                                                  border_mode=cv2.BORDER_REFLECT_101,
                                                  shift_limit=0),
                              A.VerticalFlip(p=0),
                              A.MotionBlur(blur_limit=5, p=0.8),
                              A.RandomToneCurve(scale=0.25, p=1),
                              A.ISONoise(color_shift=(0.01, 0.05),
                                         intensity=(0.1, 0.25), p=0.5),
                              A.RandomBrightnessContrast(brightness_limit=0.05,
                                                         contrast_limit=0, p=0.5)],
                              bbox_params=A.BboxParams(format='yolo'))

    return transform
except:
    raise Exception('Invalid Argument')

def writeYolo(self, coords, count, name):

    with open(f'/content/augmented/txt/{name}{count}.txt', "w") as f:
        for x in coords:
            f.write("%s %s %s %s %s %s %s %s %s %s\n" % (x[-1], x[0], x[1], x[2], x[3]))

def start(self):
    imagespath = self.images_path
    maskpath = self.mask_path
    txtpath = self.anno_path
    max_iter = self.N
    bboxes = None
    mask = None

    for filename in sorted(os.listdir(imagespath)):

        if filename.endswith(".jpg") or filename.endswith(".JPG"):
            title, ext = os.path.splitext(os.path.basename(filename))
            image = Augment.readImage(self, f'{imagespath}/{filename}')
            img = copy.deepcopy(image)

            # For synthesized image
            if self.imgtype == 'syn':
                annname = title + '.txt'
                maskname = title + '.png'
                if maskname in os.listdir(maskpath):
                    print(f'{maskpath}/{maskname}')
                    mask = cv2.imread(f'{maskpath}/{maskname}')
                if annname in os.listdir(txtpath):
                    bboxes = Augment.readYolo(self, f'{txtpath}/{annname}')
            elif self.imgtype == 'real':
                annname = title + '.txt'
                if annname in os.listdir(txtpath):
                    bboxes = Augment.readYolo(self, f'{txtpath}/{annname}')

        for count in range(1, max_iter+1):
            transform = Augment.getTransform(self)
            name = title + '(' + str(count) + ')' + '.jpg'
            if self.imgtype == 'syn':
                transformed = transform(image=img, bboxes=bboxes, mask=mask)
                transformed_bboxes = transformed['bboxes']
                transformed_mask = transformed['mask']
                cv2.imwrite(f'/content/augmented/masks/{name}', transformed_mask)
                Augment.writeYolo(self, transformed_bboxes, count, title)

```





## Appendix C: YOLOv3 Training and Evaluation Codes

```
# Evaluation
%cd /content/darknet
!./darknet detector map data/training.data cfg/yolov3_training.cfg
/drive/yolov3/yolov3_training_final.weights /content/darknet/data/obj/D2S_YOLO/test -
thresh 0.5

# Upload image from local computer to Colab
%cd /content
from google.colab import files
from os import path
import cv2
import matplotlib.pyplot as plt

uploaded = files.upload()
for name, data in uploaded.items():
    with open('image1.jpg', 'wb') as f:
        f.write(data)
print('saved file ' + name)

# Perform Inference
!./darknet detector data/training.data cfg/yolov3_training.cfg
/drive/yolov3/yolov3_training_final.weights
/content/darknet/data/obj/D2S_YOLO/test/D2S_618214.jpg -thresh 0.5

# Visualize Predicted Image
# Show Prediction of Image
def imShow(path):
    %matplotlib inline
    image = cv2.imread(path)
    height, width = image.shape[:2]
    resized_image = cv2.resize(image,(3*width, 3*height), interpolation = cv2.INTER_CUBIC)
    fig = plt.gcf()
    fig.set_size_inches(18, 10)
    plt.axis('off')
    plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
    plt.show()

imShow('predictions.jpg')
```

## Appendix D: RetinaNet Training and Evaluation Codes

```

# Environment Setup
!git clone https://github.com/bubbliiiing/retinanet-pytorch.git
%cd /content/retinanet-pytorch
!pip install -r requirements.txt

# Mount Google Drive
from google.colab import drive
drive.mount('/content/gdrive')
!ln -s /content/gdrive/MyDrive/ /drive
!ls /drive

# Remove of existing folders in dataset directory
import shutil
shutil.rmtree('/content/retinanet-pytorch/VOCdevkit/VOC2007/Annotations')
shutil.rmtree('/content/retinanet-pytorch/VOCdevkit/VOC2007/ImageSets')
shutil.rmtree('/content/retinanet-pytorch/VOCdevkit/VOC2007/JPEGImages')

# Extract D2S dataset to directory
!unzip /drive/PyTorch_RetinaNet/D2S_RetinaNet.zip -d
/content/retinanet-pytorch/VOCdevkit/VOC2007

# Copy Pretrained weights
!cp /drive/PyTorch_RetinaNet/retinanet_resnet50.pth /content/retinanet-pytorch/model_data

# Copy modified training script to directory
!cp /drive/PyTorch_RetinaNet/train.py /content/retinanet-pytorch

# Train
!python train.py!cp /drive/PyTorch_RetinaNet/retinanet.py /content/retinanet-pytorch

# Evaluation (mAP)
!cp /drive/PyTorch_RetinaNet/retinanet.py /content/retinanet-pytorch
!python get_gt_txt.py
!python get_map.py

# Check Inference Time
total_time = 0
image_ids = open('D2SDataset/ImageSets/test/test.txt').read().strip().split()
for image_id in tqdm(image_ids):
    image_path = "D2SDataset/JPEGImages/"+image_id+".jpg"
    img = Image.open(image_path)
    test_interval = 1
    tact_time = retinanet.get_FPS(img, test_interval)
    total_time +=tact_time
    avg_time = total_time/360
print(str(avg_time) + ' seconds, ' + str(1/avg_time) + 'FPS, @batch_size 1')

# Perform Inference
!python predict.py

```

## Appendix E: GAN Training Script

```

# Mount OneDrive
!curl https://rclone.org/install.sh | sudo bash
!rclone config
!sudo mkdir /content/OneDrive
!nohup rclone --vfs-cache-mode writes mount onedrive: /content/OneDrive &

# Set Environment (CycleGAN)
%cd /content
!git clone https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix.git
%cd /content/pytorch-CycleGAN-and-pix2pix
!pip install -r requirements.txt

# Extract Dataset
!unzip /content/OneDrive/CycleGAN_v7.zip -d /content/dataset

# Train CycleGAN
!python train.py --dataroot /content/dataset --name syn_2_real --model cycle_gan \
  --netD basic --n_epochs 100 --n_epochs_decay 100 --batch_size 4 \
  --netG resnet_9blocks --preprocess scale_width_and_crop --load_size 900 \
  --crop_size 256 --display_id 0 --lambda_identity 0.4 --lambda_A 8 \
  --lambda_B 8

# Generate Rendered Images
!python test.py --dataroot /content/dataset/train_syn --direction AtoB --model test
  --name syn_2_real --preprocess scale_width --load_size 800 --no_flip
  --netG resnet_6blocks --no_dropout --num_test 3000

# Export Rendered Images
from google.colab import files

!zip -r Rendered_Images_800.zip \
  /content/pytorch-CycleGAN-and-pix2pix/results/syn_2_real/test_latest/images
!cp /content/pytorch-CycleGAN-and-pix2pix/Rendered_Images_800.zip \
  /content/drive/MyDrive/Data_Synthesis

# Setup Environment (AttentionGAN)
%cd /content
!git clone https://github.com/Ha0Tang/AttentionGAN.git
%cd /content/AttentionGAN
!pip install -r requirements.txt

# Train AttentionGAN
!python train.py --dataroot /content/dataset --name syn_2_real --model attention_gan --
dataset_mode unaligned --pool_size 20 --no_dropout \
  --norm instance --lambda_A 8 --lambda_B 8 --lambda_identity 0.4
  --load_size 800 --batch_size 4 --niter 100 --niter_decay 100
  --gpu_ids 0 --display_id 0 \
  --display_freq 100 --print_freq 100 --netG ours

# Generate Rendered Images
!python test.py --dataroot /content/dataset --direction AtoB --model attention_gan
  --name syn_2_real --preprocess scale_width --load_size 800 \
  --no_flip --netG resnet_9blocks \
  --no_dropout --norm instance --phase test --gpu_ids 0 --saveDisk
  --num_threads 1

# FID Computation
!pip install pytorch-fid
!python -m pytorch_fid /content/dataset/testA /content/dataset/train_ren \
  --device cuda:0 --num-workers 0

```

## Appendix F: YOLOv5 Training Script

```
# Mount OneDrive
!curl https://rclone.org/install.sh | sudo bash
!rclone config
!sudo mkdir /content/OneDrive
!nohup rclone --vfs-cache-mode writes mount onedrive: /content/OneDrive &
!unzip /content/OneDrive/YOLOv5_Dataset_All_Scene_20220402.zip -d /content/dataset

# Setup Environment
%cd /content
!git clone -b v6.0 https://github.com/ultralytics/yolov5.git
%cd /content/yolov5
!pip install -U albumentations
!pip install opencv-python-headless==4.1.2.30
!pip install -r requirements.txt

# Train YOLOv5
!python train.py --img 512 --batch 16 --epochs 150 --data coco128.yaml --weights yolov5l.pt \
  --nosave --cache --optimizer Adam
  --hyp '/content/yolov5/data/hyps/hyp.scratch-low.yaml'

# Evaluation
!python val.py --task test \
  --weights /content/models/render.pt \
  --data coco128.yaml --img 512 --verbose --save-txt --save-json

# cAcc Computation
%cd /content
!git clone https://github.com/DIYer22/retail_product_checkout_tools.git
%cd /content/retail_product_checkout_tools
!pip install -e .

!python -m rpctool '/content/yolov5/runs/val/exp/last_predictions.json' \
  '/content/dataset/test_overlap_COCO_format/annotations/instances_test2017.json'
```

## Appendix G: Light Weight Models Architectures

### 1. MobileNet V3

```

# YOLOv5 🚀 by Ultralytics, GPL-3.0 license

# Parameters
nc: 80 # number of classes
depth_multiple: 1.0 # model depth multiple
width_multiple: 1.0 # layer channel multiple
anchors:
  - [10,13, 16,30, 33,23] # P3/8
  - [30,61, 62,45, 59,119] # P4/16
  - [116,90, 156,198, 373,326] # P5/32

backbone:
  [[-1, 1, hswish, [16, 2]], # 0-p1/2

  [-1, 1, MobileNetV3_Layer, [16, 16, 3, 2, 1, 0]], # 1-p2/4

  [-1, 1, MobileNetV3_Layer, [24, 72, 3, 2, 0, 0]], # 2-p3/8
  [-1, 1, MobileNetV3_Layer, [24, 88, 3, 1, 0, 0]], # 3

  [-1, 1, MobileNetV3_Layer, [40, 96, 5, 2, 1, 1]], # 4-p4/16
  [-1, 1, MobileNetV3_Layer, [40, 240, 5, 1, 1, 1]], # 5
  [-1, 1, MobileNetV3_Layer, [40, 240, 5, 1, 1, 1]], # 6
  [-1, 1, MobileNetV3_Layer, [48, 120, 5, 1, 1, 1]], # 7
  [-1, 1, MobileNetV3_Layer, [48, 144, 5, 1, 1, 1]], # 8

  [-1, 1, MobileNetV3_Layer, [96, 288, 5, 2, 1, 1]], # 9-p5/32
  [-1, 1, MobileNetV3_Layer, [96, 576, 5, 1, 1, 1]], # 10
  [-1, 1, MobileNetV3_Layer, [96, 576, 5, 1, 1, 1]], # 11

  [-1, 1, SPPF, [1024, 5]], # 12
  ]

head:
  [[-1, 1, Conv, [512, 1, 1]], # 13
  [-1, 1, nn.Upsample, [None, 2, 'nearest']],
  [[-1, 8], 1, Concat, [1]], # cat backbone P4
  [-1, 3, C3, [512, False]], # 16

  [-1, 1, Conv, [256, 1, 1]], # 17
  [-1, 1, nn.Upsample, [None, 2, 'nearest']],
  [[-1, 3], 1, Concat, [1]], # cat backbone P3
  [-1, 3, C3, [256, False]], # 20 (P3/8-small)

  [-1, 1, Conv, [256, 3, 2]],
  [[-1, 17], 1, Concat, [1]], # cat head P4
  [-1, 3, C3, [512, False]], # 23 (P4/16-medium)

  [-1, 1, Conv, [512, 3, 2]],
  [[-1, 13], 1, Concat, [1]], # cat head P5
  [-1, 3, C3, [1024, False]], # 26 (P5/32-large)

  [[20, 23, 26], 1, Detect, [nc, anchors]], # Detect(P3, P4, P5)
  ]

```

## 2. ShuffleNet V2

```

# YOLOv5 🚀 by Ultralytics, GPL-3.0 license

# Parameters
nc: 80 # number of classes
depth_multiple: 1.0 # model depth multiple
width_multiple: 1.0 # layer channel multiple
anchors:
  - [10,13, 16,30, 33,23] # P3/8
  - [30,61, 62,45, 59,119] # P4/16
  - [116,90, 156,198, 373,326] # P5/32

backbone:

  [[-1, 1, Conv_maxpool, [24]], # 0-P2/4

  [-1, 1, ShuffleNetV2_Layer, [116, 2]], # 1-P3/8
  [-1, 3, ShuffleNetV2_Layer, [116, 1]], # 2

  [-1, 1, ShuffleNetV2_Layer, [232, 2]], # 3-P4/16
  [-1, 7, ShuffleNetV2_Layer, [232, 1]], # 4

  [-1, 1, ShuffleNetV2_Layer, [464, 2]], # 5-P5/32
  [-1, 3, ShuffleNetV2_Layer, [464, 1]], # 6

  [-1, 1, SPPF, [1024, 5]], # 7
  ]

head:
  [[-1, 1, Conv, [512, 1, 1]], # 8
  [-1, 1, nn.Upsample, [None, 2, 'nearest']],
  [[-1, 4], 1, Concat, [1]], # cat backbone P4
  [-1, 3, C3, [512, False]], # 11

  [-1, 1, Conv, [256, 1, 1]], # 12
  [-1, 1, nn.Upsample, [None, 2, 'nearest']],
  [[-1, 2], 1, Concat, [1]], # cat backbone P3
  [-1, 3, C3, [256, False]], # 15 (P3/8-small)

  [-1, 1, Conv, [256, 3, 2]],
  [[-1, 12], 1, Concat, [1]], # cat head P4
  [-1, 3, C3, [512, False]], # 18 (P4/16-medium)

  [-1, 1, Conv, [512, 3, 2]],
  [[-1, 8], 1, Concat, [1]], # cat head P5
  [-1, 3, C3, [1024, False]], # 21 (P5/32-large)

  [[15, 18, 21], 1, Detect, [nc, anchors]], # Detect(P3, P4, P5)
  ]

```

### 3. GhostNet

```

# YOLOv5 🚀 by Ultralytics, GPL-3.0 license

# Parameters
nc: 80 # number of classes
depth_multiple: 1.0 # model depth multiple
width_multiple: 1.0 # layer channel multiple
anchors:
  - [10,13, 16,30, 33,23] # P3/8
  - [30,61, 62,45, 59,119] # P4/16
  - [116,90, 156,198, 373,326] # P5/32

backbone:

  [[-1, 1, Conv, [16, 3, 2, 1]], # 0-P1/2
   [-1, 1, GhostModule, [16, 16, 3, 1]], # 1

   [-1, 1, GhostModule, [24, 48, 3, 2]], # 2-P2/4
   [-1, 1, GhostModule, [24, 72, 3, 1]], # 3

   [-1, 1, GhostModule, [40, 72, 3, 2, True]], # 4-P3/8
   [-1, 1, GhostModule, [40, 120, 3, 1, True]], # 5

   [-1, 1, GhostModule, [80, 240, 3, 2]], # 6-P4/16
   [-1, 3, GhostModule, [80, 184, 3, 1]], # 7
   [-1, 1, GhostModule, [112, 480, 3, 1, True]],
   [-1, 1, GhostModule, [112, 480, 3, 1, True]],

   [-1, 1, GhostModule, [160, 672, 3, 2, True]], # 10-P5/32
   [-1, 1, GhostModule, [160, 960, 3, 1]], # 11
   [-1, 1, GhostModule, [160, 960, 3, 1, True]],
   [-1, 1, GhostModule, [160, 960, 3, 1]],
   [-1, 1, GhostModule, [160, 960, 3, 1, True]],
   [-1, 1, Conv, [960]],

   [-1, 1, SPPF, [1024, 5]], # 16
  ]

head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 9], 1, Concat, [1]], # cat backbone P4
   [-1, 3, C3, [512, False]], # 20

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 5], 1, Concat, [1]], # cat backbone P3
   [-1, 3, C3, [256, False]], # 24 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 21], 1, Concat, [1]], # cat head P4
   [-1, 3, C3, [512, False]], # 27 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 17], 1, Concat, [1]], # cat head P5
   [-1, 3, C3, [1024, False]], # 30 (P5/32-large)

  [[24, 27, 30], 1, Detect, [nc, anchors]], # Detect(P3, P4, P5)
  ]

```

## Appendix H: Tables

Table K - 1: State-of-Art Models for Retail Product Checkout

Author	Dataset applied	State-of-Art Model	Train	Test	Configuration	mAP (%)	Precision	Recall	IoU	Run time (ms)
Liu et al. (2019)	Self-captured (Bread)	Faster R-CNN (VGG-16)	452	3	Steps: $70 \times 10^3$	99.14				540
Koturwar, Shiraishi and Iwamoto (2019)	Self-captured (Groceries)	Faster R-CNN (ResNet-101)	20000	300	Steps: $200 \times 10^3$ Learning rate: 0.003		0.93	0.84	0.86	
				600			0.84	0.98	0.85	
				100			0.61	0.84	0.78	



Rigner (2019)	Self-captured (Groceries)	Mask R-CNN (ResNet-50)	2430			72.3				590
		YOLOv3				59.9				127
		RetinaNet (ResNet-50)				71.8				257
Ning, Li and Ramesh (2019)	D2S Dataset	Mask R-CNN (ResNet – 101)	2880	360	Learning rate: 0.002 Epochs : 10 Optimizer: SGD	0.788				
		Mask R-CNN (ResNet – 50)				0.753				

Wu et al. (2016)	Web-crawled	YOLO + CaffeNet	YOLO 63271 CaffeNet 317593		YOLOv1 Iterations: $30 \times 10^3$	66.4 (One)				69.63 75
					CaffeNet Iterations: $100 \times 10^3$ Learning rate: 0.0005 Batch Size: 128 Momentum: 0.9	65.7 (Two)				
						64.1 (Three)				
Oh and Chun (2020)	Self-captured (Beverage)	YOLOv3	2800		Epochs: 16200 Learning rate: 0.001 Momentum: 0.9 Weight decay: 0.0005	82.28				

Xie, Wang and Zhao (2021)	RPC Dataset	Faster R-CNN (ResNet-101)		6000		96.98				
		YOLOv3				82.32				
		RetinaNet (ResNet-101)				99.56				