UNIVERSITY OF NAIROBI

SCHOOL OF COMPUTING AND INFORMATICS

Convolutional Neural Network to enhance stock taking

by

Lilian Awuor Onyango

P52/11882/2018

Supervised by Dr Wanjiku Nganga

Research project report submitted in partial fulfilment of for the award of Master of Science in Computational Intelligence

Declaration

Researcher's declaration

The research, as presented in this report, is my original work and has not been presented for any award in any other university

Signed ……………………………………… Date…………………………………

Lilian Awuor Onyango

P52/11882/2018

Supervisor's approval

This project report has been submitted in partial fulfilment for the requirements of the award of the Degree of Master of Science in Computational Intelligence in the University of Nairobi with my approval as the university supervisor.

Signed……………………………………… Date……………………………………

Dr. Wanjiku Nganga

School of Computing and Informatics

University of Nairobi

Acknowledgments

I am grateful to God for the health and energy granted to me to enable me carry out my research. I sincerely thank my supervisor Dr Wanjiku Nganga for the guidance she has given me through course of the project enabling me to improve and refine my work. I am grateful to my family for the encouragement and support they have granted me throughout my studies.

Abbreviations

CNN	Convolutional neural networks
DSP	Digital signal processing
IRI	Inventory record inaccuracy
MM	Material management
RFID	Radio frequency identification
YOLO	You only look once
NIN	Network in network
SVM	Support vector machine
ANN	Artificial neural network
N^4	Neural network nearest neighbour
DPM	Deformable parts model
AP	Average precision
FPS	Frames per second
mAP	Mean average precision
G-OPS	Generalized operations per second
BOW	Bag of words
LBP	Local binary pattern
HOG	Histogram of oriented gradient
GPU	Graphical processing unit

Table of Contents

List of figures

List of tables

Abstract

In the current competitive economic environment, there is a lot of focus in optimization of processes and providing high quality customer experience. This study explores the use of deep learning particularly convolutional neural network to enhance the retail store stock taking process. It provides a review of literature on different convolutional neural network architectures to identify the best fit for image object detection and count. It highlights some of the image analysis applications in various sectors such as counting fish, yield estimation and construction site management.

YOLO is noted to be perform well based on the literature review and the study further implements and compares the performance of YOLO v2 and YOLO v3 in object detection and count. The implementation leverages on the pretrained weights on ImageNet and further training is done on open data image set of retails stores. Both YOLOv2 and YOLOv3 achieve mean average precision above 75%, however YOLOv3 is leading attaining a mean average precision of 81.86%.

Keywords convolutional networks, YOLO, stock take

CHAPTER 1: INTRODUCTION

1.1. Background

Data has been recognized as one of the key drivers of business advantage and organisations are consciously defining their data use cases, making investments in data to achieve informed decision making, operational efficiencies and ultimately be able to value data as an asset. The initial focus in data analysis has been on the structured data generated throughout the internal business processes which gives a good descriptive story of progress over time and is one of the key inputs into development of predictive models. In addition to structured data, significant research has been done towards use of unstructured data, both text and images, to gather insights and provide services such as facial recognition security feature on smart phones, targeted advertising based on history search, automate quality control using image analysis in manufacturing plants and automatic parking fee clearance based on number plate images amongst others.

Stock take is a key process in inventory management and retail analytics. It guides the business on when to reorder goods, which items to stop stocking, what items are fast moving and identify any loss of goods. Currently, stock takes in retail stores are done manually at a set interval. This process is very manual, tedious and prone to many errors. Stock take tends to be done at closing time or off-peak hours when there is little movement at the store (Kamali, 2018). The support staff are distributed across the store to manually count and record the volume of the different products on a tallying sheet. These sheets are then collected by the supervisor to be later captured in the system for onward processing. This process flow has two key points of weakness. The repetitive count of products and the transfer of data from tallying sheets into the system. These are monotonous tasks which are highly prone to error impacting the accuracy of data captured on stock levels (Kull, et al., 2013).

Digital image processing is a subarea in digital signal processing (DSP) that has been in existence since the 1960s and 1970s when digital computers become available. However, implementation of DSP was limited to critical operations such as oil exploration, medical images and space exploration since computers were expensive. Since then various techniques have been developed for digital signal processing the greatest catalyst being the growth in the computational power, data storage, display and transmission (Bhausaheb Shivajirao Shinde, 2011).

Application of digital image processing has increased significantly with implementations such as posture detection (Wang, et al., 2016), automatic reading of coordinates from an image and linking the image to the specific location on the map (Goodfellow, et al., 2014) and emotion detection through analysis of facial expressions (Luoh, et al., 2010).

Deep learning is one of the key techniques applied in digital image analysis particularly convolutional neural network to enable image classification, object detection, image retrieval, semantic segmentation and human pose estimation (Guo, et al., 2015). The convolutional neural networks have been preferred in image analysis owing to its ability to better manage the data volume and ability to share parameters reducing the number of variables to tune.

1.2. Problem statement

Stock take is a mandatory exercise to confirm level of stock for accounting purposes as well as support in deciding when to reorder and restock the shelves. Currently this exercise is done manually, requires a lot of resources depending on the size of the store and is prone to errors. Automated stock take would enable retail store to optimize their stock process and free time to address other needs in the store. Retail store managers are aware of the importance of the being informed on accurate stock levels, however they are still faced with a backlog of inaccuracies they are not able to resolve (Ishfaq & Raja, 2019). Manual inventory count is highly repetitive such that the concentration span of an individual reduces drastically making it difficult to obtain accurate results (Neeley 1983, 1987) hence the need to automate this process.

1.3. Main Objective

The study aims to implement image object detection and count using convolutional neural networks for stock take in a supermarket.

1.4. Specific Objective

1. To review the processes and challenges in stock taking in a retail store
2. To review existing literature on convolutional neural network applied in digital image processing
3. To implement a prototype convolutional neural network to enable object detection and count in retail product image
4. To measure the effectiveness of the convolutional neural network in image object detection and count for stock take

1.5. Significance

As retails stores open up to more channels of transactions and collaborate with third party ecommerce platforms, the need to ensure product availability is critical to meet customer demands (Kamali, 2018). Ensuring optimal stock levels is required to avoid the financial impact of low or very high stock levels. High stock levels require additional expenditure on storage and potential loss in damage or spoilage. Whereas low stock levels can lead to high delivery cost due urgency to restock. Central to management of stock level is being able to determine the current stock count in time to make a decision on when to reorder considering the lead time to delivery.

The automated stoke count using images aims to increase accuracy and improve on time spent in determining the stock levels.

1.6. Scope of study

The analysis focuses on one category of inventory, bottled soda. However, the prototype solution presented in the paper can be applied in count of other bottled products. It is applicable on the products whose quantities are based on count and not weight.

The next section, chapter 2, provides a review of existing literature on stock take and convolutional neural networks. It gives an overview of the stock process and the challenges noted. On convolutional neural network, it shows the development in the technique, use of this technique in different domains and its performance. Chapter 3 gives an overview of the methodology applied in developing the convolutional neural network for image analysis in retail shelves stock take. Chapter 4 outlines the performance of the convolutional neural network and chapter 5 highlights the conclusion and recommendations based on the study.

CHAPTER 2: LITERATURE REVIEW

2.1 Review of inventory count

Materials management (MM) is an essential business activity responsible to maintain a flow of materials and other stock items (inventory) used in an organization, and deals with other functions in supply chain management, such as the planning, acquisition, and supply of overall material flows throughout the whole supply chain. One of the key roles of MM is inventory control. The main reason that an organization holds inventories is to ensure to satisfy customers demands at the right time (Kull, et al., 2013).

Accuracy in inventory level has been a challenge over time and various technologies have been implemented to automated this process. One of these technologies is barcoding. This has largely been used in the checkout process. Barcodes are a set of vertical bars printed on products by the manufacturer. The vertical bars store basic information about the product such as name, weight and colour. Using a bar code reader at the checkout point, the product can be identified and the appropriate price charged and stock level adjusted at the same time.

Another technology that has been applied in inventory management is the radio frequency identification (RFID). Use of RFID has increased over time however it still remains an expensive option that may not be applicable in certain setups(Lee & Özer., 2007). It uses near field technology such that the scanner is able to pick multiple RFID signals from a given distance.

Accuracy of inventory count is affected by both transaction dependent elements and transaction independent elements. The transaction dependent are replenishments, misplaced items, product returns, incorrect deliveries, partial deliveries, frequency of replenishments while the transaction independent factors are internal product movement, operating multiple channels, theft, spoilage or damage (Kull, et al., 2013). Order fulfilment and inventory replenishment are quite susceptible to such problems (Lee and Özer 2007).

Transaction dependent factors	**Transaction independent factors**
frequency of replenishment	internal product movement
orders and demand	theft or spoilage
misplaced items in the store	multiple channels with one store of operation
incorrect deliveries	
product returns	

Table 1: Factors affecting inventory accuracy, source (Kull, et al., 2013) (Guo, et al., 2015)

There is need for adequate inventory policies and frequent inventory count to enable quick identification of the inaccuracy in inventory level to enable better inventory management. The ability to accurately determine how much inventory is on hand is critical. This becomes the guideline on when to reorder goods to minimize the cost of holding high inventory volume or too little inventory such that the store is not able to meet customer demand. Inventory record inaccuracy is a key issue across multiple industries affecting the accounting process, mergers and acquisitions amongst others. Most stores do not know the accurate inventory value and would generally tend to hold more stock to meet the demands rather than fall short (Barratt, et al., 2010).

2.2 Deep learning

Artificial intelligence refers to algorithms that aim to mimic human behaviour. Machine learning is a subset of artificial intelligence which entails analysing data to determine the current relationships and capture this in a model that can then be applied on new instances to predict the output. Deep learning is a subset of machine learning that is inspired by the functioning of the human brain. It takes in data and works on deriving the features through layers of connected neurons. The features extracted at each layer is then transferred to the consequent layer till the last year where the prediction in computed (Guo, et al., 2015). Deep learning has gained popularity given the increased volumes of data and computational capacity such that large models can be designed and trained within shorter periods. This reduces the time taken to go through the iterative process of data collection, data preparation, training, testing and optimization.

Various deep learning algorithms such as AlexNet, GoogleNet and YOLO, have been developed to enable image processing and have been optimized to reduce the computation cost as much as possible considering the fact that a single image contains multiple features and the volume of data may grow exponentially within the neural network if not checked.

Digital image processing entails converting a continuous signal such as sound or light into a discreet format that can be used for further computational processing. An example is an image captured using a camera is basically a reflection of light signals from the object of interest which is digitized into a sequence of number to represent the light intensity at different points of the object. This sequence of numbers can then be stored and referenced to recreate the image for viewing. A two-dimensional function, f(x, y) is used to represent the images such that x and y are spatial (plane) co-ordinate, and the amplitude f is the gray level of the image at that point. Discrete values of x, y and the amplitude f make an image a digital image (Bhausaheb Shivajirao Shinde, 2011). Numerous mathematical procedures and algorithms have been developed and tested to enable processing of a wide spectrum of images in the medical domain, geospatial domain and even in commercial products such as smart phones and laptops. Some of the objectives set out in digital image processing include image enhancement to highlight certain features of interest, colour enhancement, creating augmented reality, image overlays, object detection and recognition.

2.3 Convolutional neural networks

Convolutional neural networks are widely used in image analysis due to its ability to take in a large number of inputs and reduce the dimensions systematically, managing the number of parameters to be trained and improving general performance. This is achieved when a number of parameters share weight reducing the number of parameters to be trained and the local connectivity enables learning the correlation among pixels. Below is an illustration of the building blocks of a convolutional neural network. It consists of the image input, convolution layers, pooling layers and fully connected layers.

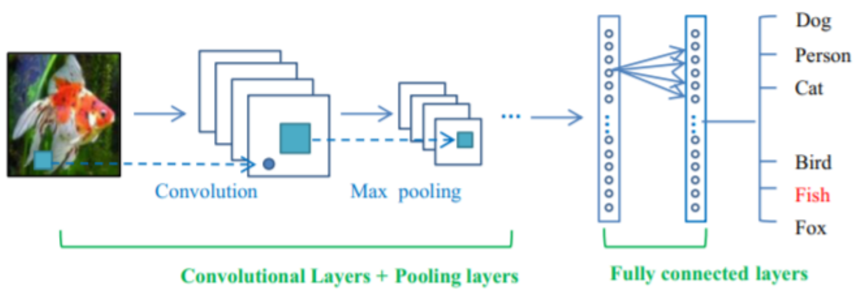


Figure 1: A general pipeline for a CNN, source (Guo, et al.,2015)

Convolutional neural networks (CNNs) has been applied in object detection, semantic segmentation, image classification, image retrieval and human pose estimation (Guo, et al., 2015).

Various architectures have been developed over time and refined to improve on performance. Some of the common architectures include AlexNet, GoogleNet, VGG -16 and 19, Network in Network, Inception v3 and ResNet. Research has been done to compare performance of these models looking at accuracy, computation cost and efficiency of the model, (Canziani, et al., 2016). Inception 3 was noted to have the highest top accuracy. Other architectures such as VGG-19 were noted to have a fairly good accuracy however its computation cost was noted to be highest as well. This led to the need to look at a third measure information density, (Canziani, et al., 2016) where GoogLeNet was noted to perform best.

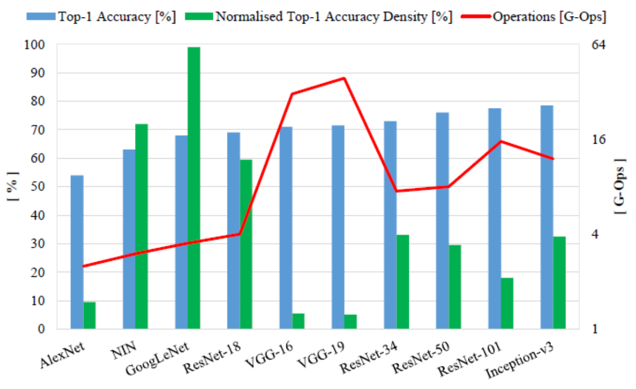


Figure 2: performance review of deep neural network architectures (Canziani, et al., 2016)

2.4 You only look once (YOLO) detection system

YOLO network architecture has been built for image detection and classification. It was inspired by the GoogLeNet model which was noted to have the best performance in previous studies. In this approach, object detection is considered as a single regression problem from image pixels to bounding box coordinates and class probabilities. As the name suggests, an image is scanned once to predict the objects in it unlike other algorithms that split the image into sections and review it section by section. YOLO is fast, this being attributed to its simple structure. YOLO takes in information of the whole image and is able to encode contextual information (Redmon, et al., 2016). At a high level, the image below demonstrates how a single convolutional network is able to predict multiple bounding boxes and classes.

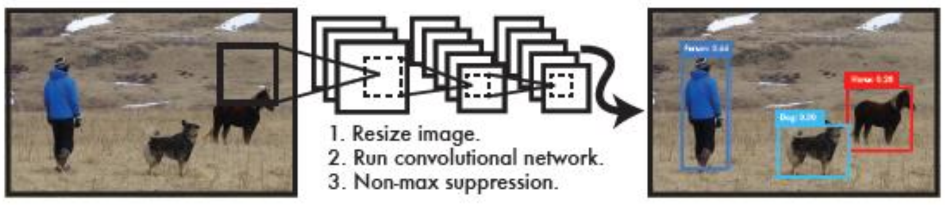


Figure 3 Illustration of YOLO image object detection and classification, source (Redmon, et al., 2016)

The first step in the analysis is to divide an input image into N x N grid. If the centre of an object falls withing that grid, that grid is used to detect the object. Each grid cell is associated with a bounding box and confidence score. The confidence score is a measure of the probability of an object existing in that frame and the degree of overlap, intersection over union, between the predicted box and the ground truth. The output of each bounding box has 5 predictions x, y, w, h, and confidence. The center of the box relative to the edges of the grid cell is captured by the x and y coordinates. The width and height are predicted relative to the whole image while the confidence prediction represents the overlap (IOU) between the predicted box and any ground truth box. In classification problem, and additional output would be the class prediction. The YOLO approach has been evaluated on the

PASCAL VOC detection dataset which has a variety of image classes using 24 convolutional layers followed by 2 fully connected layers as indicated below, (Everingham, et al., 2015).

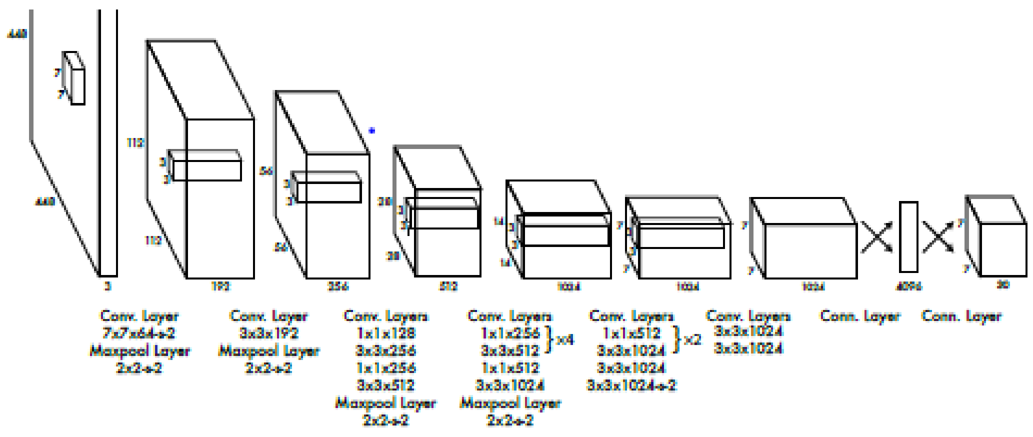
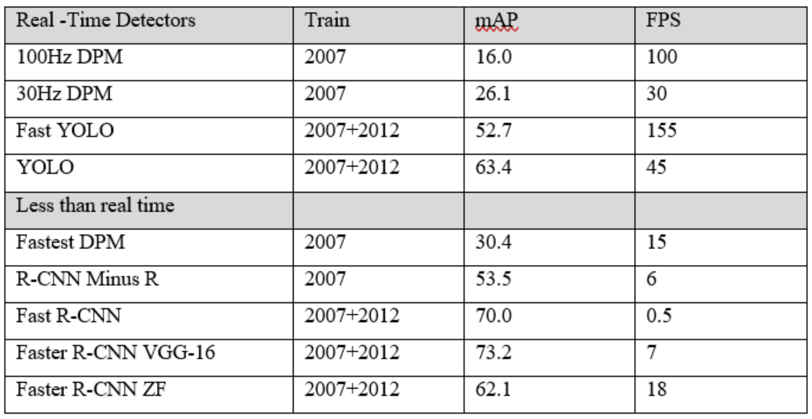


Figure 4: YOLO network architecture with output accuracy of 88%

YOLO can process images at a high speed however it faces spatial constraint limiting the number of clustered objects that the model can predict and does not perform as well in detecting small objects in a cluster (Redmon, et al., 2016). This a key areas that lead into development of advanced version of YOLO to YOLO v2, YOLOv3, YOLOv4 and currently YOLO v5.

The YOLO approach has been compared against other image analysis techniques such as the deformable parts (DPM) and R-CNN. The DPM uses a sliding window approach (P. F. Felzenszwalb, 2010) which is a disjoint pipeline that is trained independently. R-CNN applies region proposals to detect objects. This involves a combination of selective search, CNN to extract features and SVM to score the boxes, linear model to adjust the bounding boxes and non-max suppression to eliminate duplicate detections. These components are also trained independently. Compared to R-CNN, YOLO predicts fewer bounding boxes per image and has a comprehensive architecture that is jointly optimised (Redmon, et al., 2016). Below is a summary comparing performance of YOLO to DPM and R_CNN object detection systems.



Real -Time Detectors	Train	mAP	FPS
100Hz DPM	2007	16.0	100
30Hz DPM	2007	26.1	30
Fast YOLO	2007+2012	52.7	155
YOLO	2007+2012	63.4	45
Less than real time			
Fastest DPM	2007	30.4	15
R-CNN Minus R	2007	53.5	6
Fast R-CNN	2007+2012	70.0	0.5
Faster R-CNN VGG-16	2007+2012	73.2	7
Faster R-CNN ZF	2007+2012	62.1	18

Table 2: Comparison of YOLO to DPM and R-CNN (Redmon, et al., 2016)

Yolo was noted to be the fastest detector on record for PASCAL VOC detection and was twice as accurate as any other detector. A detailed error analysis was done for R_CNN and YOLO to better understand the root cause. It was noted that for R_CNN most of the errors were background errors while for YOLO it was localization error (Redmon, et al., 2016)

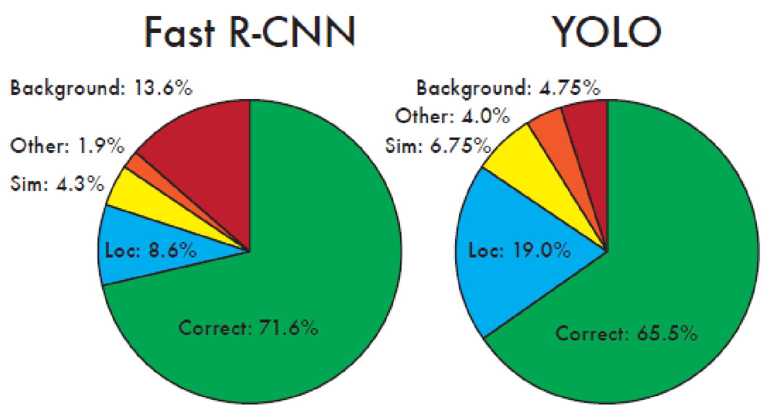
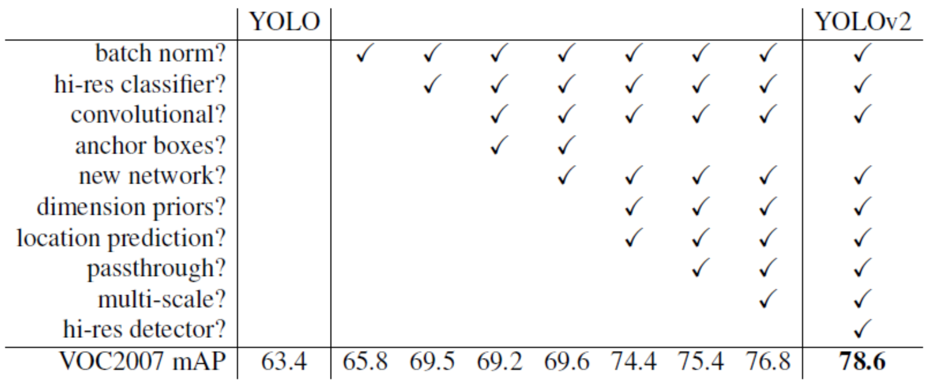


Figure 5:Analysis of errors (Redmon, et al., 2016)

Further enhancements have been made on YOLO giving rise to YOLOv2 and YOLO9000. The focus was to improve on localization error and recall in YOLO. This study went further to increase the type of objects that can be detected leveraging on the classes available in image classification taking noted that labelling images for detection is more intensive compared to labelling for classification or tagging.

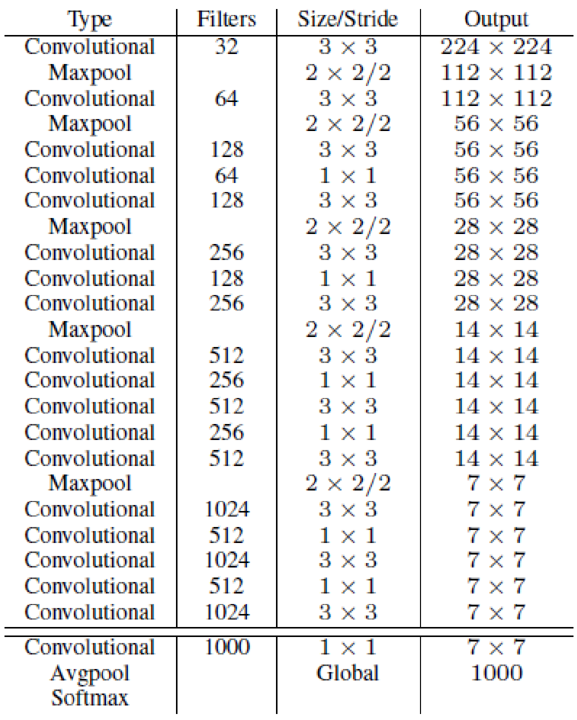
In YOLOv2 the enhancement has been made on normalization, use of K-means to determine the initial box dimensions, multiscale training, use of anchor boxes and high-resolution classifiers as captured below.



	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Figure 6 : Enhancements to YOLO (Redmon, et al., 2016)

In YOLO9000, the team developed a new architecture called the Darknet-19 leverage on the Network in Network model. It consists of 19 convolutional layers and 5 max pooling layers while the YOLO architecture consists of 24 convolutional neural networks and 2 fully connected layers.



Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Table 3: Darknet -19 architecture (Redmon, et al., 2016)

The Darknet – 19 achieved to increases the number of objects that can be detected to over 9000 objects leveraging on the labelled data available from ImageNet with labels pulled from WordNet which enables hierarchical classification.

2.5 Related work

2.5.1 Classifying equipment at a construction site using convolutional neural networks

Construction site project managers are faced with the challenge of ensuring maximum output on each day. There is need to ensure the project timelines are on track and that resources are being fully utilised to enable sharing of these resource across multiple sites. To track the site operations, one of the approaches is to use the Global Positioning system however this cannot be attached to every gadget on site. This led to exploration of computer vision to analyse the utilisation of the equipment and safety at the site, (Soltania, et al., 2017). The first step in application of computer vision was being able to accurately detect the equipment the surveillance footage.

The study compared conventional classifiers and convolutional neural networks to determine the better model. The conventional classifiers tested include Local Binary Pattern (LBP), Histogram of Oriented Gradient (HOG) and bag of words (BoW). The study focused on a subset of constructions equipment which are excavators, loaders and dump track. Amongst the conventional classification approaches, the HOG was leading with an accuracy of 47% followed by LBP with an accuracy of 41% and lastly BOW with an accuracy of 38%. The classification was again tested using deep neural network models particularly the AlexNet-SVM, AlexNet and VGG-f. The accuracy of the models was compared and noted that AlexNet-SVM was leading with an accuracy of 83% followed by AlexNet with an accuracy of 78% and VGG-f attained an accuracy of 68%. The CNN based methods obtained better performance compared to the conventional methods. The analysis was further reviewed and it was noted that the synthetics images used in training appeared rich in colour and brand new unlike the actual equipment which due to wear and tear tends to have a dark brown colour. This difference in the training set and reality on the ground was a key factor in the performance noted.

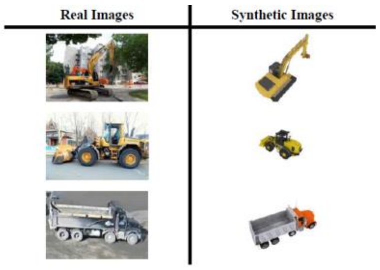
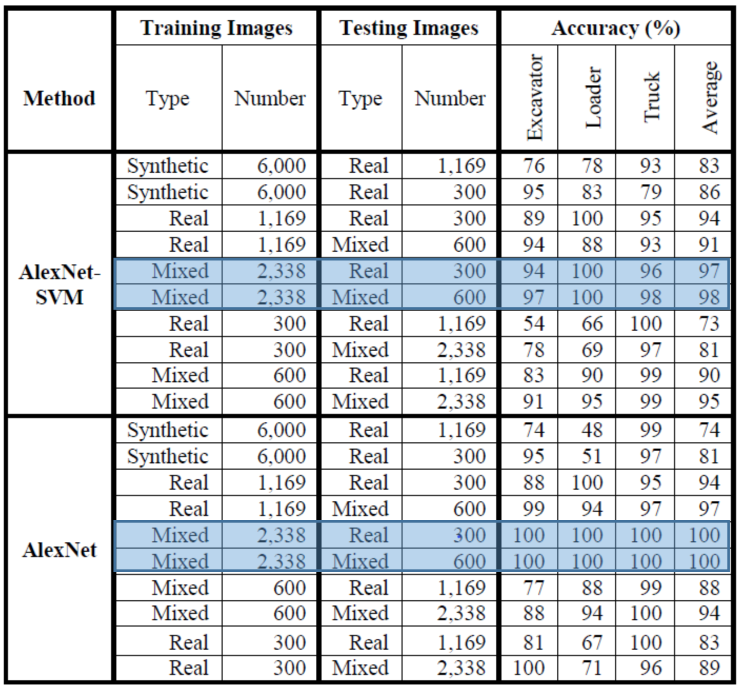


Figure 7: sample of real and synthetic image, (Soltania, et al., 2017)

The study further analysed the performance of the AlexNet-SVM and AlexNet using different scenarios of synthetic and real images as indicated below. Based on the results obtained, addition of synthetics images to the training set improved the training results and the accuracy of the model, (Soltania, et al., 2017).



Method	Training Images		Testing Images		Accuracy (%)			
	Type	Number	Type	Number	Excavator	Loader	Truck	Average
AlexNet-SVM	Synthetic	6,000	Real	1,169	76	78	93	83
	Synthetic	6,000	Real	300	95	83	79	86
	Real	1,169	Real	300	89	100	95	94
	Real	1,169	Mixed	600	94	88	93	91
	Mixed	2,338	Real	300	94	100	96	97
	Mixed	2,338	Mixed	600	97	100	98	98
	Real	300	Real	1,169	54	66	100	73
	Real	300	Mixed	2,338	78	69	97	81
	Mixed	600	Real	1,169	83	90	99	90
	Mixed	600	Mixed	2,338	91	95	99	95
AlexNet	Synthetic	6,000	Real	1,169	74	48	99	74
	Synthetic	6,000	Real	300	95	51	97	81
	Real	1,169	Real	300	88	100	95	94
	Real	1,169	Mixed	600	99	94	97	97
	Mixed	2,338	Real	300	100	100	100	100
	Mixed	2,338	Mixed	600	100	100	100	100
	Mixed	600	Real	1,169	77	88	99	88
	Mixed	600	Mixed	2,338	88	94	100	94
	Real	300	Real	1,169	81	67	100	83
	Real	300	Mixed	2,338	100	71	96	89

In addition to the application of CNN in identifying excavators, loaders, and dump trucks there is need to add classification of other equipment (Soltania, et al., 2017).

2.5.2 Application of convolutional neural networks in counting fish

One of the global sustainable development goals focuses on life below water to ensure that the ecosystem in protected. Institutions have been set up to continuously monitor the fishing activities looking at the species and size of fish harvested. A study has been done to explore application of computer vision to help track the number, species and size of fish harvested as captured in the surveillance videos (French, et al., 2015) . This will increase the efficiency in enforcing the fishing policies.

The first step in the analysis was to carry out the foreground segmentation separating the actual fish from the background and irrelevant objects such as conveyer belt and people. This was achieved using CNN approaches specifically the neural network nearest neighbour (N^4) fields algorithm. At this point the foreground pixels of fish are separated to be counted and later on classified into the different species.

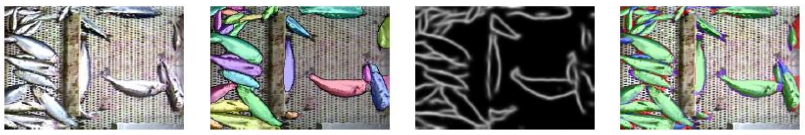


Figure 8 Steps from the original image to box prediction, edge detection and finally measure accuracy (French, et al., 2015)

A number of factors were to be considered while building the model. The model was to be applied on a video stream where there would be long period of no activity and periods when a lot of fish is to be reviewed. The video is also captured as other fishing activities are proceeding, so a lot of human hands would be captured in the video. In addition, the fish would overlap, have different sizes and be randomly positioned. With all these factors considered, the model was able to reduce the error range to 2% to 16%, (French, et al., 2015).

2.5.3 Convolutional neural networks to count fruits

The agricultural sector is one of the industries that is aggressively integrating technology into its operations with an aim to ensure returns on investment. Being able to accurately determine the expected yield would be highly beneficial in planning and in the future potentially a trusted collateral to be able to obtain financing. Manual yield estimation in time consuming and tedious, this study aimed to explore the application of computer vision to count tomatoes in a big farm, (French, et al., 2015).

Keeping in mind the target working environment, the model to be developed had to address issues such as lighting variation, overlap and scale variations. The team sort to explore object count without necessarily detecting the object by focusing on the colour variations. Tomatoes would be a shade of red while the leaves would be green.

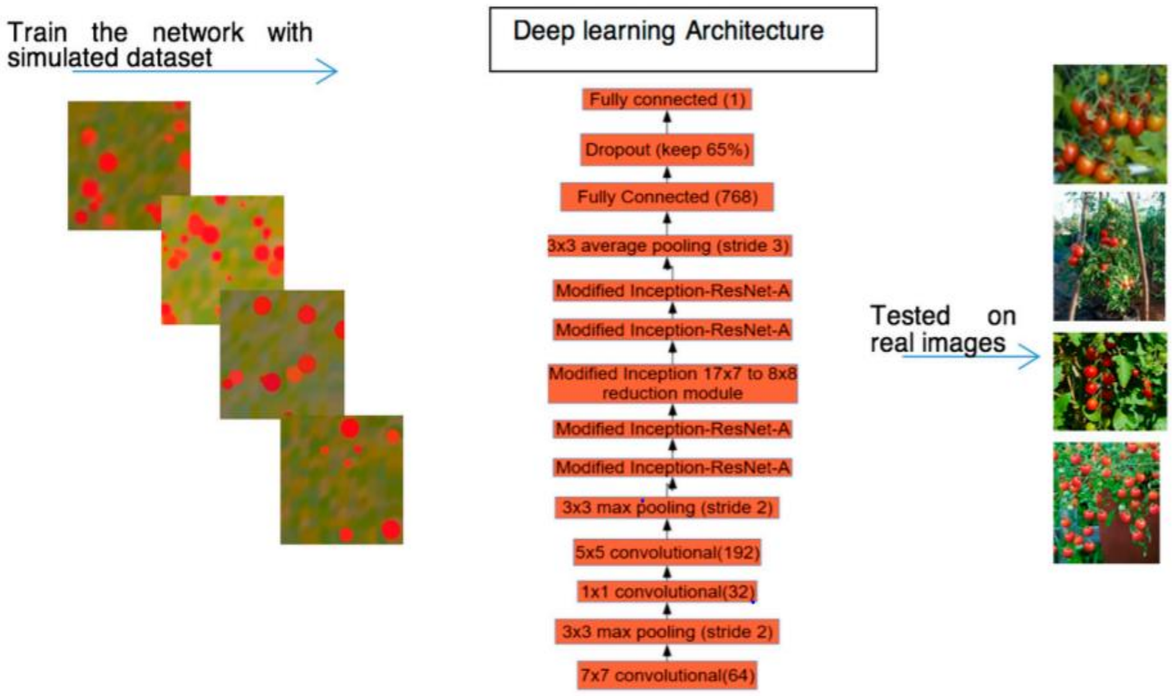

Train the network with simulated dataset
Deep learning Architecture
Fully connected (1)
Dropout (keep 65%)
Fully Connected (768)
3x3 average pooling (stride 3)
Modified Inception-ResNet-A
Modified Inception-ResNet-A
Modified Inception 17x7 to 8x8 reduction module
Modified Inception-ResNet-A
Modified Inception-ResNet-A
3x3 max pooling (stride 2)
5x5 convolutional(192)
1x1 convolutional(32)
3x3 max pooling (stride 2)
7x7 convolutional(64)
Tested on real images

Figure 9 framework for counting tomatoes using CNN, source (Rahnemoonfar & Sheppard, 2017)

The study was able to apply CNN, a modified inception-Resnet and simulate based training where the model was trained using synthetic data and tested on real images achieving an accuracy of 91%. The approach was able to address illumination, occlusion and scale variations.

R	P	GT	R	P	GT	R	P	GT	R	P	GT
	36	38		27	24		18	17		27	28
	22	25		21	23		15	14		12	12
	22	22		13	12		14	14		14	13
	20	25		19	19		38	39		16	16
	22	22		16	17		16	19		24	24

Figure 10 sample of image, predicted count and ground truth, source (Rahnemoonfar & Sheppard, 2017)

The algorithm performs fairly well in counting fruits whose colour is a shade of red, it did not cover the green fruits, unripe fruits. This can readily be extended by adding the green fruit examples in the training dataset, (Rahnemoonfar & Sheppard, 2017). In terms of delivery of the solution to the farmers, the solution can be packaged as a mobile app however aerial images with a surveillance camera would be ideal.

2.6 Challenges in image processing

In carrying out image processing the technique applied needs to consider the contrast between object and background, degree of object clustering, object texture and its variations, object size and its variations, complexity of the objects among others (Barbedo, 2012).

Researchers have developed different techniques to minimise the impact of these challenges as seen in the YOLO architecture where the architecture involves analysis of the images at different scales so that its able to analyse objects of different sizes. The approach applied in YOLO was noted to have fewer background errors compared to the R-CNN model and YOLO2 tried to improve on the localization error through incorporating K-means clustering to automatically find good bounding boxes priors.

2.7 Gap

Convolutional neural networks are leading in the space of image analysis. Various architectures have been proposed and tested on the key computer vision datasets, Pascal VOC dataset, to benchmark and compare performance. The YOLO algorithm was noted to perform well in comparison to other algorithms. Further analysis on the false positives indicate that YOLO faces challenges in localization and identification of highly clustered images. Later versions of YOLO have been released with a key focus to improve on the challenges of YOLO version 1 and also expand the classes of objects being detected.

Convolutional neural network has been applied in various areas such as counting fish in fisheries, monitoring work at a construction site and counting apples in a farm and crowd counting. This study explores the application and performance of the YOLO convolutional neural network in count of soda in supermarket shelves. This will further train the algorithm on highly clustered objects and increase the classes of objects that can be detected.

CHAPTER 3: RESEARCH DESIGN AND METHODOLOGY

3.1 Introduction

This chapter highlights how the research on convolutional neural networks for stock take has been structured. It covers the research design of the study, the data collection process, model development and evaluation metrics.

3.2 Research design

Quantitative research entails collecting and converting data into numerical form to enable statistical calculations, inferencing and draw conclusions (Bryman, 2004). This study is a quantitative research. It aims to identify an appropriate convolutional neural network architecture to enable image object count for stock take. The images obtained are transformed into numerical data and relationships to enable object detection and provide a count as an output. The research evaluates the YOLO v2 and YOLO v3 on object detection and counting of soda bottles in a retail store.

The key phases in building the neural network entailed data collection and preparation, building the model followed by iterative steps of training and evaluating the model.

3.3 Data collection and preparation

The YOLO algorithm is applied as a form of supervised learning requiring an input of images with predefined bounding boxes to enable it learn the class of images to be detected. It requires a set of retail images with annotation indicating the class, bonding box and position of the image as indicated below. This is done using the BBox label tool to generate the bounding boxes and python code to convert the labels into the format:

<class id> <Xo/X> <Yo/Y> <W/X> <H/Y>

where,

class id = label index of the class to be annotated

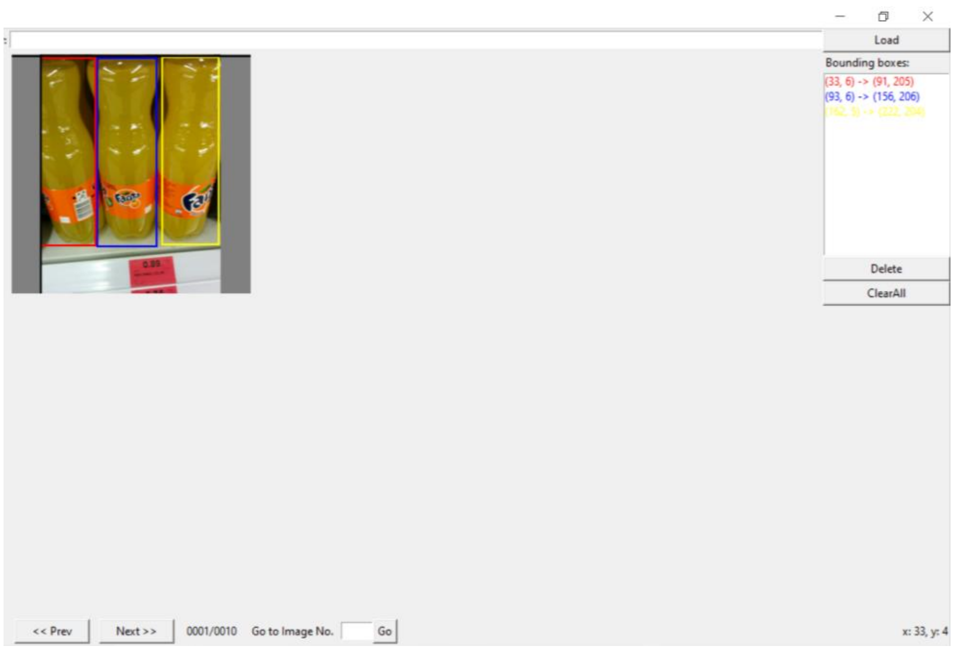
Xo = bounding box’s center, x coordinate

Yo = bounding box’s center, y coordinate

H = bounding box height

X = image width

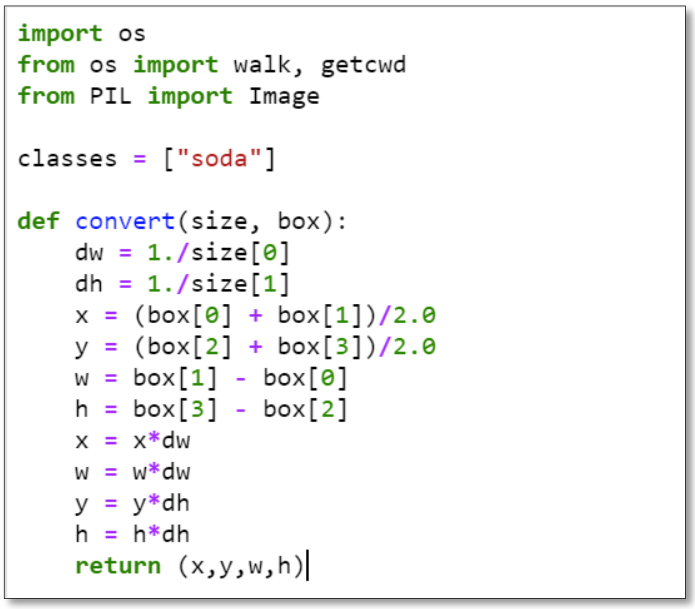
Y = image height



```
(33, 6) -> (91, 205)
(93, 6) -> (156, 206)
(162, 3) -> (222, 204)
```

Figure 11: Illustration of image annotation using BBOx labelling tool

The BBOx tool enables labelling of multiple objects within the image capturing the dimensions of the bounding box as displayed in the top right. The dimensions captured is the start and end position of the height and width of the image. This further adjusted using python code to convert it into the YOLO format [category number] [object center in X] [object center in Y] [object width in X] [object width in Y] which provides dimensions relative to the dimensions of the image.



```python
import os
from os import walk, getcwd
from PIL import Image

classes = ["soda"]

def convert(size, box):
    dw = 1./size[0]
    dh = 1./size[1]
    x = (box[0] + box[1])/2.0
    y = (box[2] + box[3])/2.0
    w = box[1] - box[0]
    h = box[3] - box[2]
    x = x*dw
    w = w*dw
    y = y*dh
    h = h*dh
    return (x,y,w,h)
```

Figure 12: Function to convert bounding box labels to yolo format

The image and the dimensions of the bounding boxes are saved in one location as pair with the same file name as illustrated below.

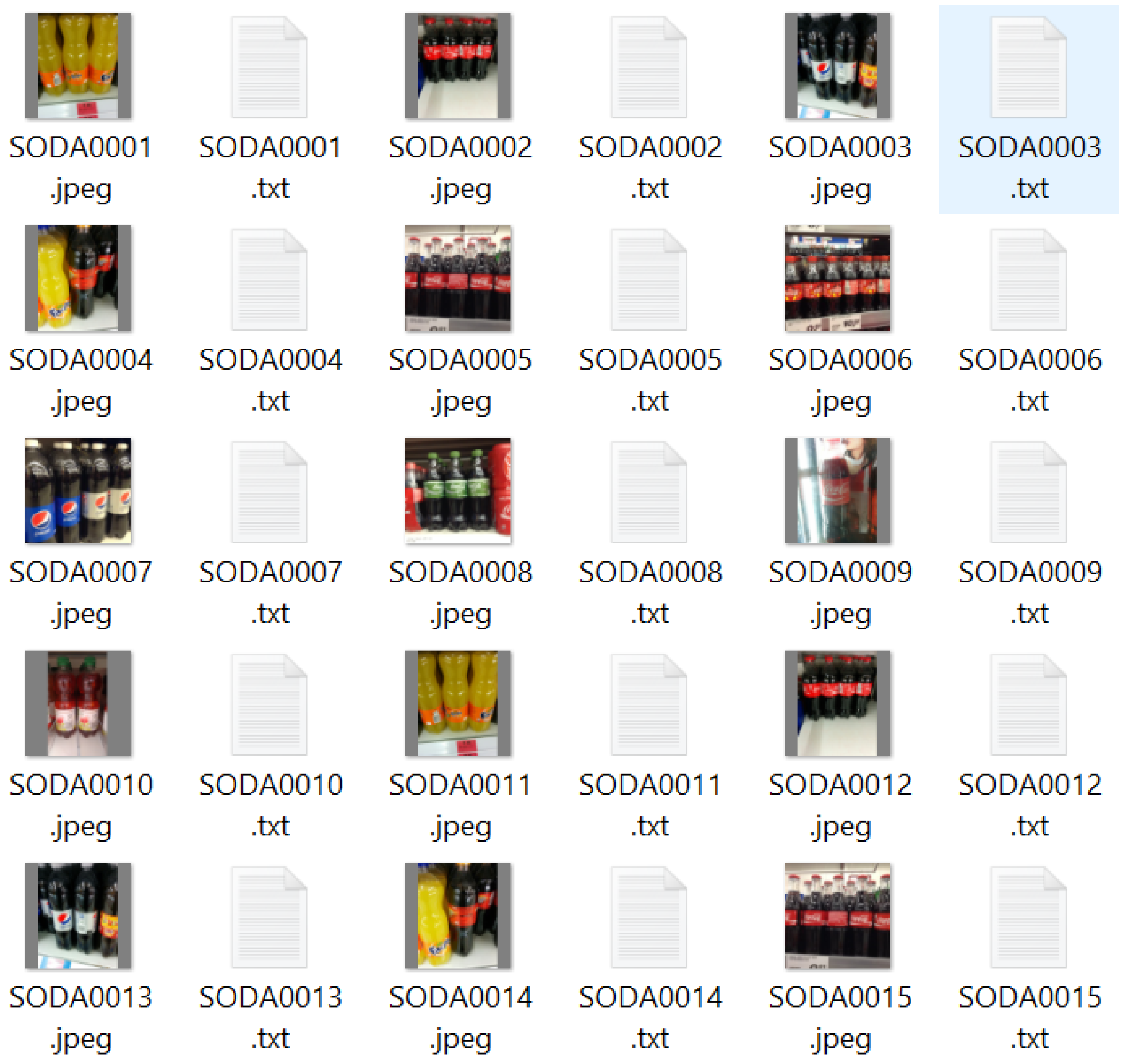
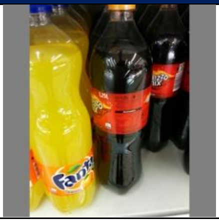


Figure 13: Sample model training files

	Sample image	Bounding box labels generated <class id> <Xo/X> <Yo/Y> <W/X> <H/Y>
1.		0 0.29296875 0.494140625 0.3203125 0.96484375 0 0.58984375 0.49609375 0.21875 0.9453125
2.		0 0.49609375 0.5 0.3125 0.984375

3.		0 0.30078125 0.515625 0.1953125 0.75
		0 0.51171875 0.513671875 0.203125 0.76171875
		0 0.724609375 0.513671875 0.19921875 0.76171875

Table 4: sample bounding box labels generated

The neural network is trained and tested based on the Freiburg groceries dataset, focusing on the soda category. 500 images are loaded and labelled using the BBOx tool providing a total of 1,300 bounding box examples. This then randomly split into 400 images for training and 100 images for testing. This was equivalent to 1,301bouding boxes for training and 294 bounding boxes for testing.

3.4 Development environment

The darknet neural network framework is used to build the convolutional neural network and train the model. This is an open source platform that provides the foundation to build the YOLO convolutional neural network. This framework is built in C.

Google colab environment is used for building, training and testing the model. It is a free cloud service provided by Google. It provides access to a single Tesla T4 GPU which enables faster training of the model. This is an online environment and is integrated with google drive to provide storage. The google colab notebook is used as the code editor.

A local Anaconda python environment is setup on a single user machine running on windows to enable the data pre-processing. This is used to setup the BBOX tool and convert bounding box labels to the YOLO format.

3.5 Neural network design and development

A single neural network is applied on the full image. When the image is loaded it is converted into a matrix with a series of numbers ranging from zero to 255. To enable easier computation and inferencing, the matrix is normalized by diving by 255 to get number between 0 and 1 to represent the different shades of red, blue and green that form the picture. The image is further resized to the requirements of the model to be a cubic image. Below is a sample image representation in the model.

```
[[[[0.22745098 0.22352941 0.20392157]
   [0.22745098 0.22352941 0.20392157]
   [0.22745098 0.22352941 0.20392157]
   ...
   [0.36862746 0.40392157 0.43137255]
   [0.37254903 0.40784314 0.43529412]
   [0.37254903 0.40784314 0.43529412]]

  [[0.22745098 0.22352941 0.20392157]
   [0.22745098 0.22352941 0.20392157]
   [0.22745098 0.22352941 0.20392157]
   ...
   [0.44313726 0.47843137 0.5058824 ]
   [0.43529412 0.47058824 0.49803922]
   [0.43529412 0.47058824 0.49803922]]

  [[0.22745098 0.22352941 0.20392157]
   [0.22745098 0.22352941 0.20392157]
   [0.22745098 0.22352941 0.20392157]
   ...
   [0.5294118  0.5647059  0.5882353 ]
   [0.5176471  0.5529412  0.5764706 ]
   [0.5019608  0.5372549  0.5647059 ]]

  ...
```

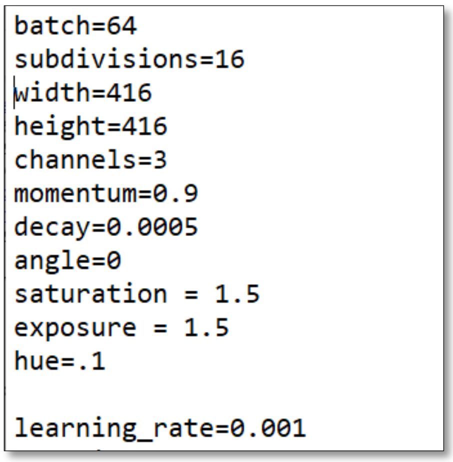
Shape of the image is (1, 416, 416, 3)

- 1 image
- 416 x 416 x 3, 3 channels to capture red, blue and green (RGB) shades.

Initialization of parameters

They key purpose of a neural network is to tune the hyperparameters to derive a pattern that enables us predict the output of unseen instances. In this case the hyperparameters to be tuned are the weights, learning rate, and bias.

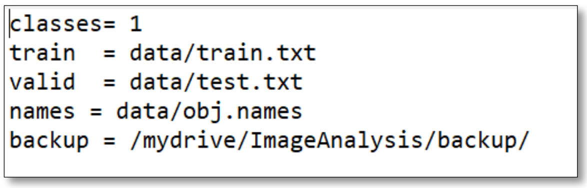
The YOLOv2 and YOLOv3 model is loaded and initialized. The learning rate is set at 0.001, decay is set at 0.0005, iterations set at 2000 and momentum set at 0.9. The bias and weights are initialized to the YOLOv2 and YOLOv3 pretrained weights respectively.



```
batch=64
subdivisions=16
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
```

Figure 14: Sample of the YOLO configuration file

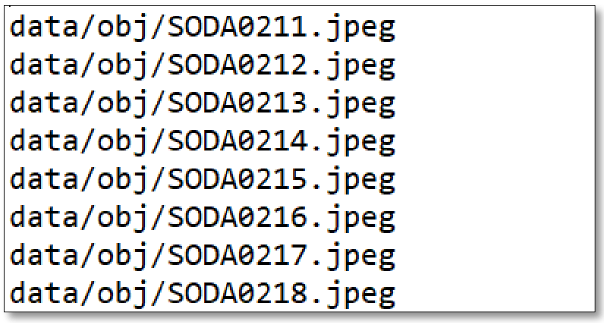
The next step is to define the number of classes, the path to the training set file, the test set file, the class labels and the location to save the trained weights.



```
classes= 1
train  = data/train.txt
valid  = data/test.txt
names = data/obj.names
backup = /mydrive/ImageAnalysis/backup/
```

Figure 15: Sample data configuration file

The train.txt file and the test.txt file contains the path to the images to be loaded into the model as captured below.

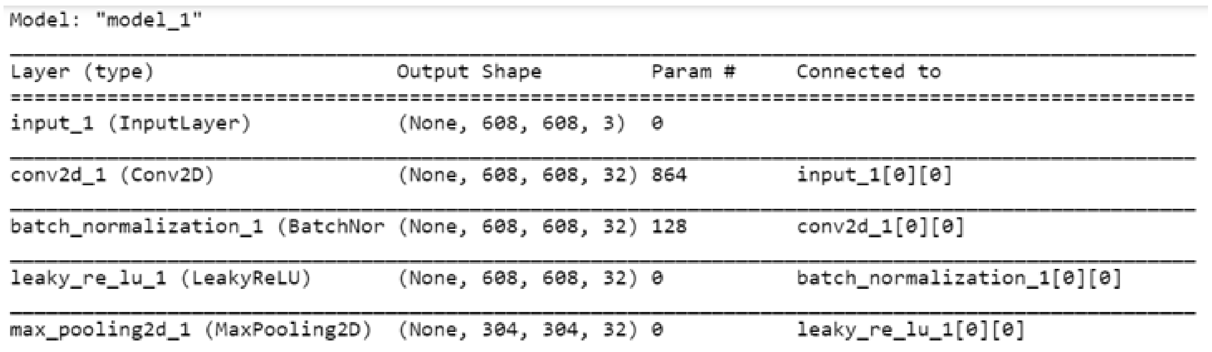


```
data/obj/SODA0211.jpeg
data/obj/SODA0212.jpeg
data/obj/SODA0213.jpeg
data/obj/SODA0214.jpeg
data/obj/SODA0215.jpeg
data/obj/SODA0216.jpeg
data/obj/SODA0217.jpeg
data/obj/SODA0218.jpeg
```

Figure 16: Sample training file configuration,train.txt.

Iterative training

The convolutional neural network training begins with a forward propagation. The input image matrix is feed into the first convolutional layer which consists of the padding, convolution process, normalization, activation function leaky ReLU and max pooling as indicated below. The output of this first layer then becomes the input for the next layer.



```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 608, 608, 3)  0
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 608, 608, 32) 864         input_1[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 608, 608, 32) 128         conv2d_1[0][0]
__________________________________________________________________________________________________
leaky_re_lu_1 (LeakyReLU)       (None, 608, 608, 32) 0           batch_normalization_1[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 304, 304, 32) 0           leaky_re_lu_1[0][0]
```

The image is padded differently at each point in the neural network to either maintain the size of the input image referred to as same padding or reduce the size of the image, referred to as valid padding. This enables the neural network to maintain high performance when tasked with images of different sizes as well incorporate the information at the edge of the image.

Each layer has a varying size and number of convolutional filters followed by the batch normalization and leaky ReLU activation function. The max pooling is applied to reduce the noise and as well as the size of the output.

This analysis is done through the 23 convolutional layers of the YOLOv2 and 73 convolutional layers for the YOLOv3. The initial iteration is based on the pretrained weights of YOLO which provides a more informed start.

At the end of the first iteration, the cost function is computed and this is used to initiate the backpropagation. Back propagation involves computation of derivates with an aim to determine the appropriate change required on the weights taking into account the learning rate set. The back propagation provides the required update to the weights and bias.

The neural network then goes through another forward propagation using the new weights and bias, the cost is computed, then back propagation to update the weights. This is repeated through 2000 iterations as defined in the initialized parameters.

The output of the iterative training process are the weights for the model that will be used to carry out the image object detection and count for stock take.

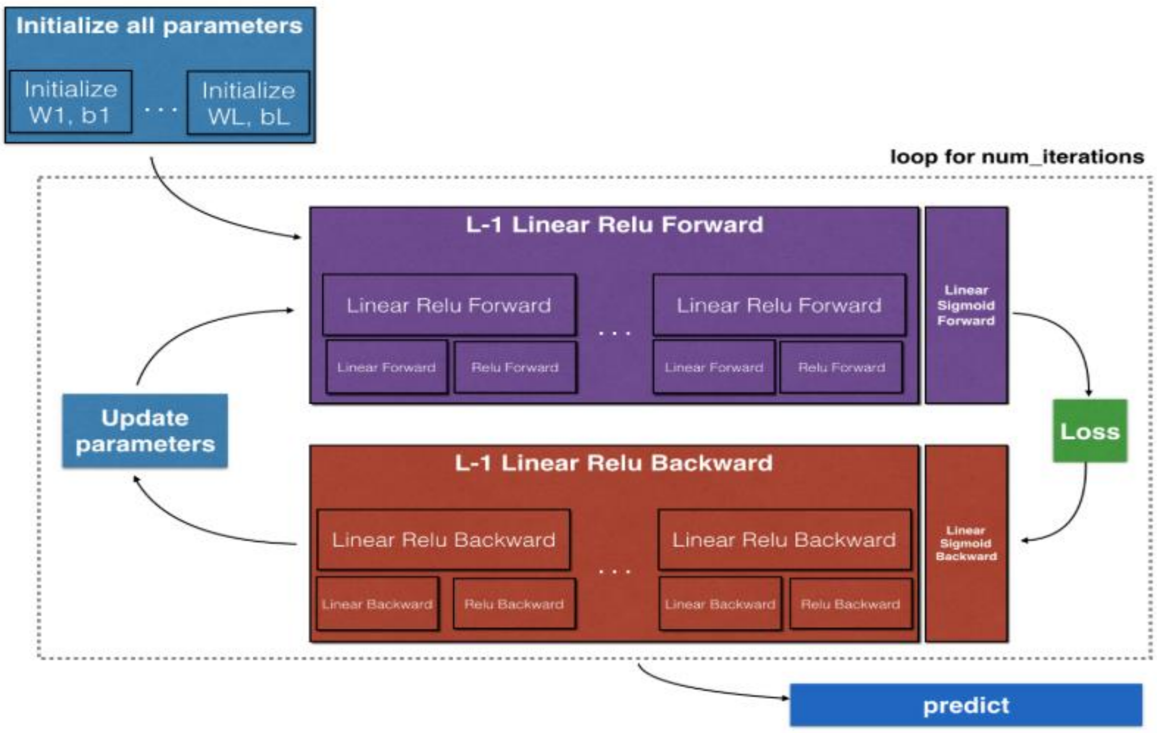


Table 5: Overview of a neural network process (Andrew NG)

3.6 Model evaluation

Image object detection is affected by occlusion, size, aspect ratio, visibility of parts, viewpoint, localization error, confusion with semantically similar objects and background (D. Hoiem, 2012). Some of the approaches applied in evaluating performance of object detection algorithms include analysis of false positive, precision-recall curves and average precision.

False positive is when detections do not correspond to the target category while localization error is when an object is predicted with a misaligned bounding box. Based on the comparison on the Fast - RCNN and YOLO, background and localization error was noted to be the largest error respectively. In evaluating this convolutional neural network, the target is to minimize the cost function and also attain high precision.

The model is evaluated by comparing the predicted count from the model and the actual count as indicated below

$$pa(\%) = \left[1 - \frac{|pc - ac|}{|ac|}\right] \times 100$$

Figure 17 Formula for percentage accuracy, source (Rahnemoonfar & Sheppard, 2017)

In addition to the percentage accuracy, the model is evaluated by the loss function which looks at the localization error, confidence loss and classification loss. The loss function is computed as indicated below.

$$
\begin{aligned}
&\lambda_{\textbf{coord}} \sum_{i=0}^{S^2}\sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} (x_i-\hat{x}_i)^2 + (y_i-\hat{y}_i)^2 \\
&+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2}\sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(\sqrt{w_i}-\sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i}-\sqrt{\hat{h}_i}\right)^2 \longrightarrow \boxed{\text{Localization error}} \\
&+ \sum_{i=0}^{S^2}\sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i-\hat{C}_i\right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2}\sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i-\hat{C}_i\right)^2 \longrightarrow \boxed{\text{Confidence}} \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c\in\text{classes}} (p_i(c)-\hat{p}_i(c))^2 \longrightarrow \boxed{\text{Class probability}}
\end{aligned}
$$

Figure 18: YOLO loss function (Redmon, et al., 2016)

CHAPTER 4: RESULTS AND DISCUSSION

4.1 Introduction

This section highlights the findings of the research. It provides an outline of the performance of the YOLO v2 and YOLO v3 in image object count for stock take. It provides an analysis of the average loss and mean average precision attained by the two models as well as an illustration of the output of the model on a sample of images.

4.2 Model results

The YOLOv2 and YOLO v3 model are trained on a single Tesla T4 GPU taking note of the average loss and mean Average precision. The average loss improves from 30.26 to 0.4181 for Yolo v2 and from 14000 to 0.3538 for Yolo v3.

YOLO struggles to get the boxes perfectly aligned with the object. In the past YOLO struggled with small objects. However, with the new multi-scale predictions YOLOv2 and YOLOv3 have relatively high mean average performance, (Redmon, et al., 2016). Below is a summary of the average loss and mAP observed during the training.

	YOLOv2		YOLOv3	
Training iterations	Average loss	mAP	Average loss	mAP
200	6.02	n/a	14000	n/a
400	1.87	n/a	2.12	n/a
600	1.12	n/a	1.31	n/a
800	0.51	n/a	1.01	n/a
1000	0.39	71%	0.98	78%
1200	0.37	76%	0.52	80%
1400	0.34	72%	0.43	78%
1600	0.21	77%	0.41	82%
1800	0.13	76%	0.32	79%
2000	0.17	76%	0.3	80%

Table 6: YOLOv2 and YOLOv3 training evaluation

The YOLOv2 model achieves a best mean average precision of 76.53% and average loss of 0.17 after 2000 iterations. The average loss is below the acceptable minimum loss of 2 and provides the weights that can be applied in the retail stock count.

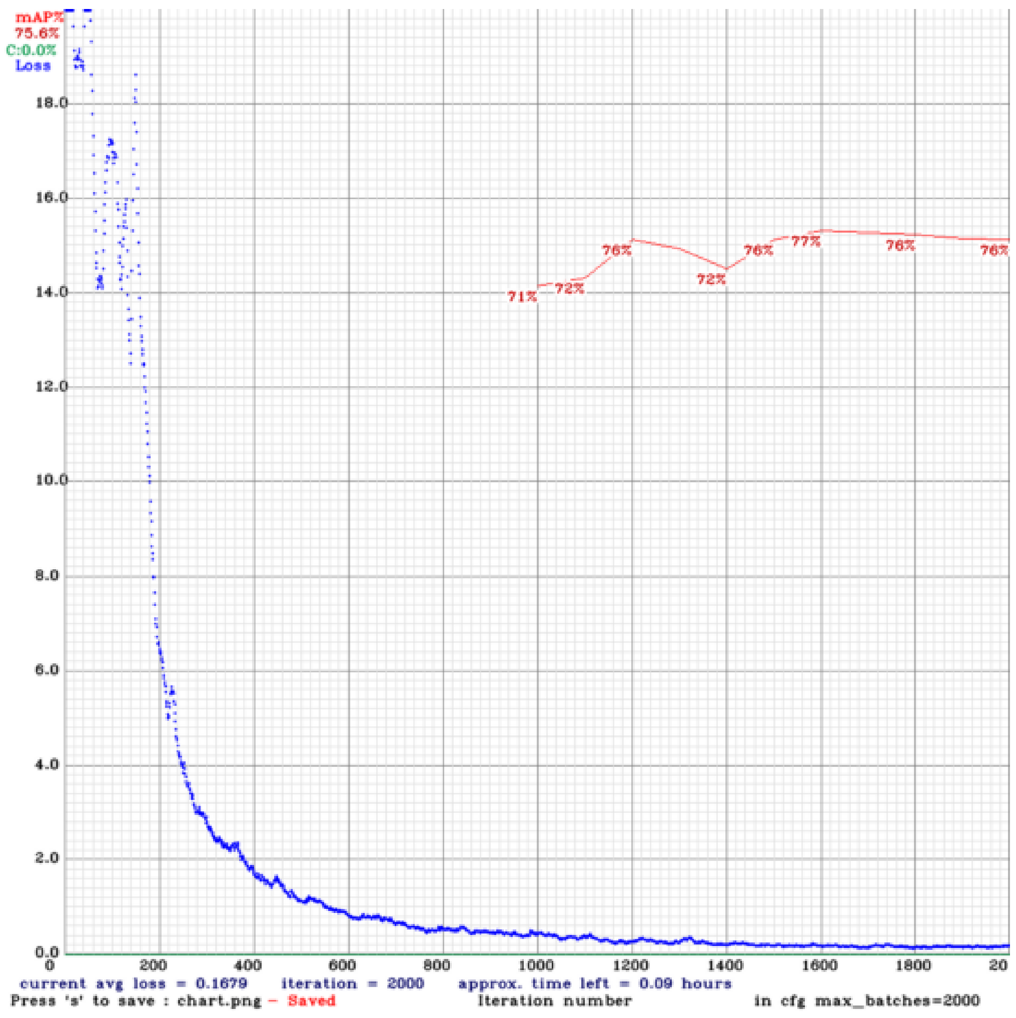


Figure 19: Yolo v2 performance

The YOLOv3 model achieves a best mean average precision of 81.86% and average loss of 0.3 after 2000 iterations. The average loss is below the acceptable minimum loss of 2 and provides the weights that can be applied in the retail stock count.

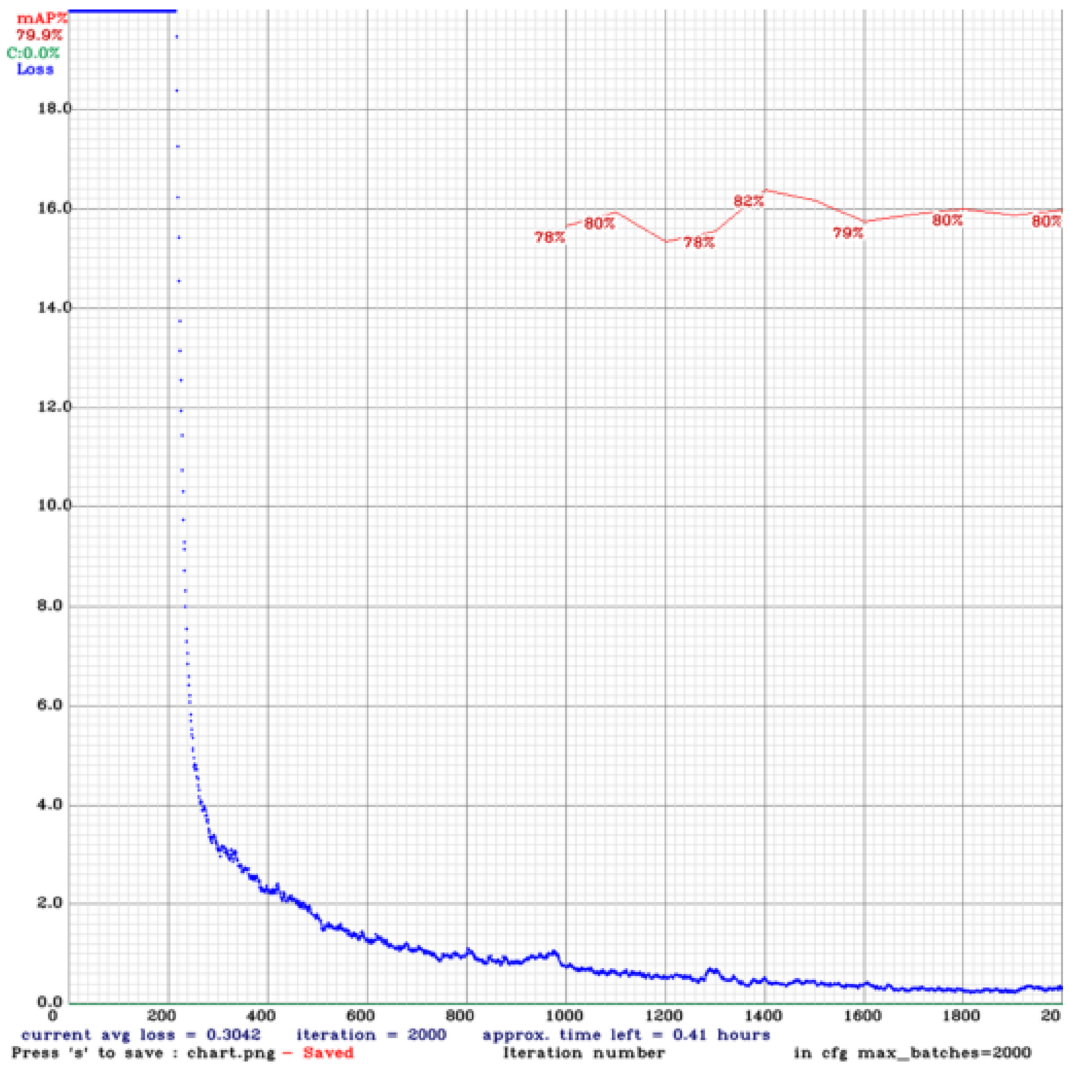
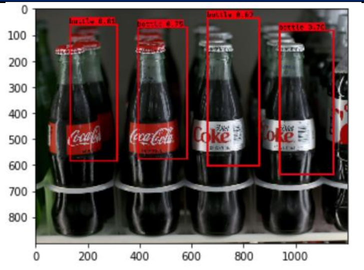
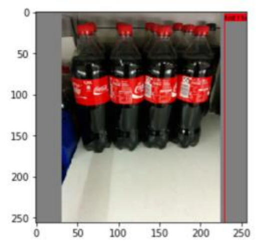
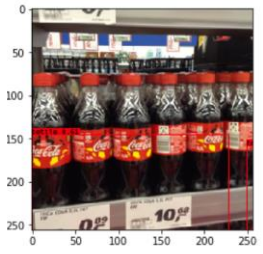
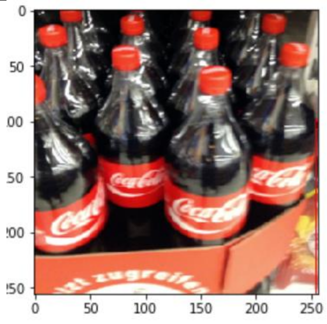
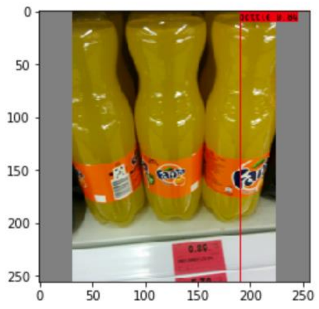
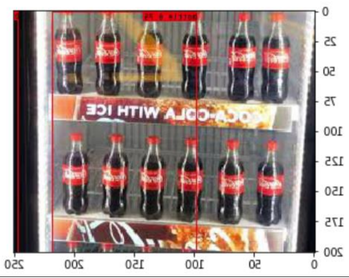
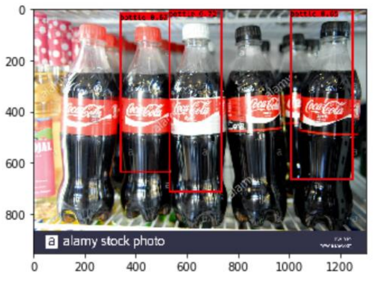


Figure 20: Yolo v3 performance 2000 iterations

4.4 Sample image tests on YOLO v2

Below is summary of object count for a sample of 15 images using the trained YOLOv2 weights.

	Image	Actual count	Count based on YOLO v2	Accuracy
1.		4	4	100%

2.		4	4	100%
3.		6	3	50%
4.		9	1	11%
5.		3	1	33%
6.		12	10	83%
7.		5	3	60%

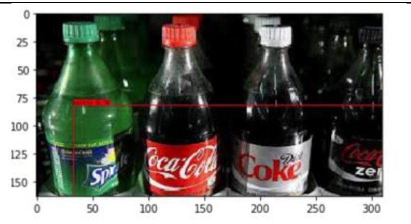
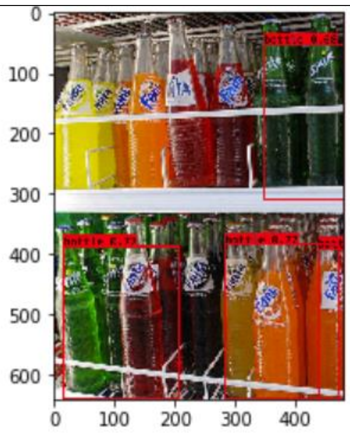
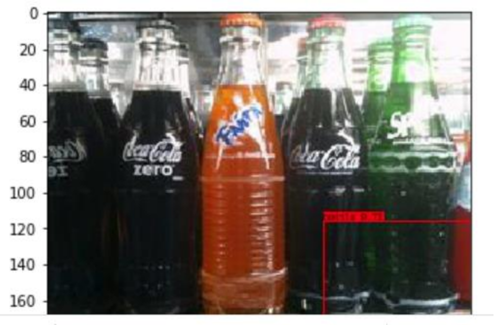
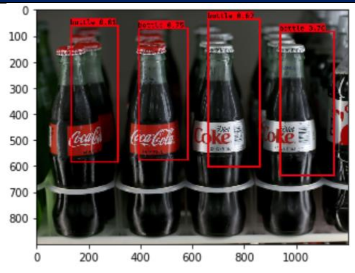
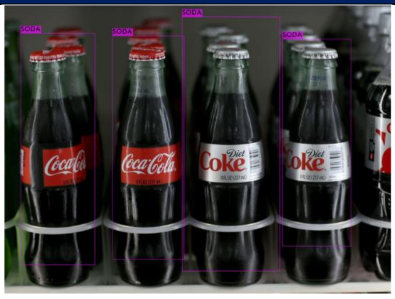
8.		3	3	100%
9.		12	10	83%
10.		5	3	60%

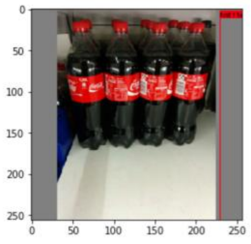
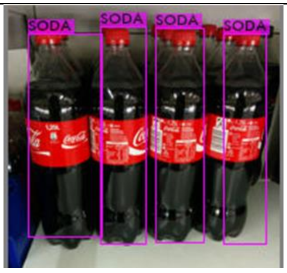
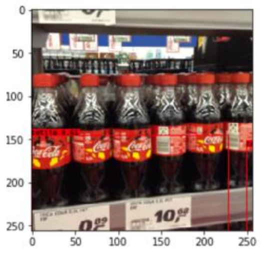
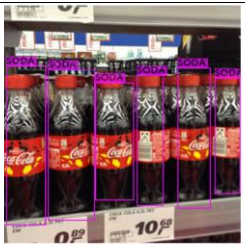
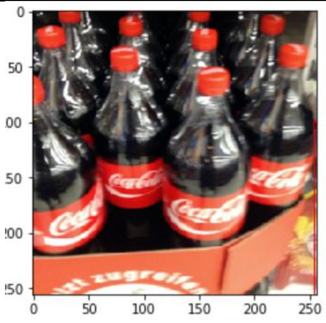
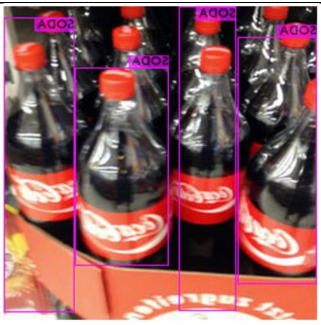
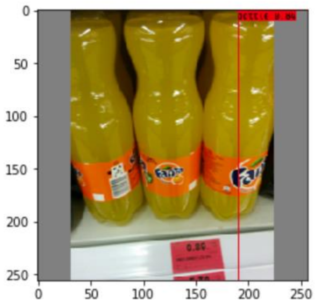
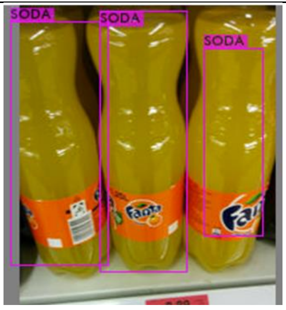
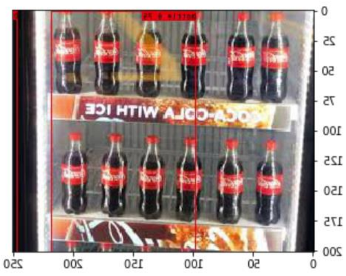
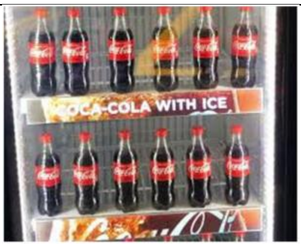
Table 7: YOLOv2 testing on sample images

Based on the model evaluation, an average accuracy of 68% is attained with the lowest score being noted when the angle of the image is close to a top aerial view as indicated in example 4 and 5. In example 10, one image is not detected due to the similarity with the black background.

The same set of images were tested using the trained YOLO v3 weights and the results are as captured below.

4.5 Sample image tests on YOLO v3

	Yolo v2	Yolo v3	Actual count	Accuracy
1.	 4	 4	4	100%

2.	 4	 4	4	100%
3.	 3	 6	6	100%
4.	 1	 4	9	44%
5.	 1	 3	3	100%
6.	 10	 10	12	83%

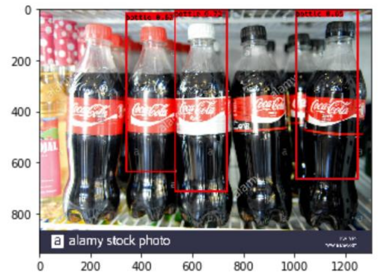
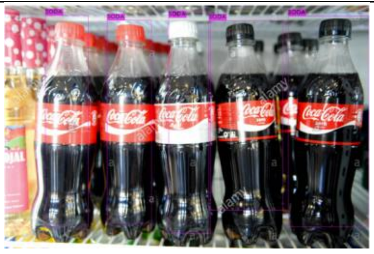
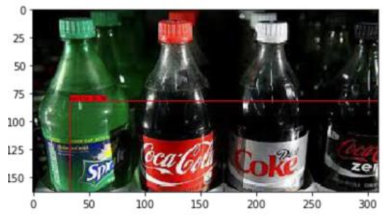
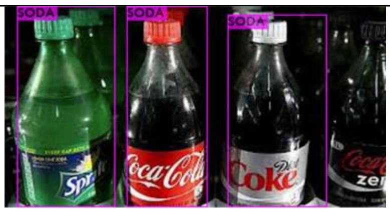
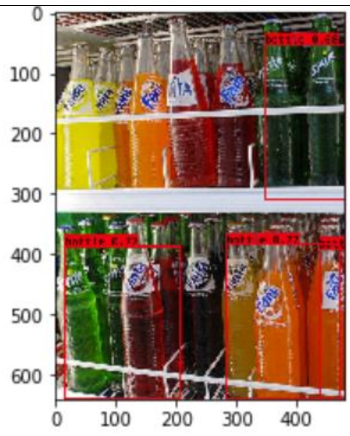
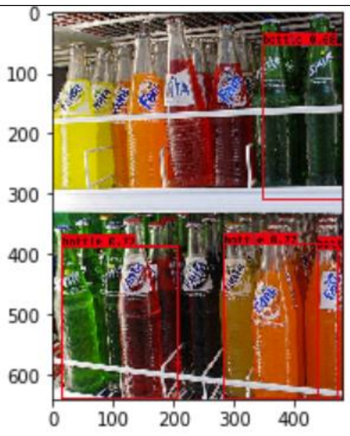
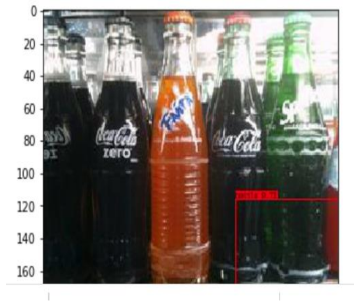
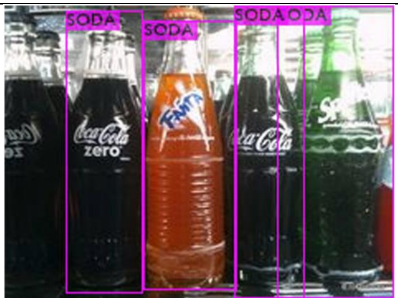
7.	 3	 5	5	100%
8.	 3	 3	3	100%
9.	 10	 10	12	83%
10.	 3	 4	5	80%

Table 8: YOLOv3 testing on sample images

Both YOLOv2 and YOLOv3 achieve a high mean average precision with YOLO v3 leading at 81.86%. In the previous evaluation of the YOLO model on the PASCAL VOC dataset YOLO v2 achieved a mAP of 76.8% while YOLOv3 achieved a mAP of 79.26%, (Ju, et al., 2019). The current training and evaluation of YOLO for retail stock take remains the same for YOLOv2 while YOLOv3 has improved by 2.6%.

CHAPTER 5: CONCLUSION

5.1 Summary of the study

The overall objective of the study was to implement a convolutional neural network to enhance stock take. This objective was accomplished by designing, implementing and testing the YOLOv2 and YOLOv3 convolutional neural network. The end product is a convolutional model able analyse images, detect and count objects using the YOLO model.

The first objective was to review the processes and challenges in stock taking in a retail store. This was achieved through literature review on store management. The study highlights the factors that impact the accuracy of stock levels and some of the technologies that have been explored to improve the process.

The second research objective of the study was to carry out a literature review of the convolutional neural network architectures and identify an appropriate convolutional neural network for image object count in a retail store. Based on the review, the YOLO architecture was noted to be leading in performance and accuracy and was select as the approach to be implemented and tested.

The third objective was to implement a prototype convolutional neural network based on the YOLO architecture. This was achieved by implementing the YOLO v2 and YOLO v3 architecture based on the darknet framework.

The fourth objective was to test the effectiveness of the convolutional neural network in image object count. This was analysed through review of the loss function and the mean average precision of the model on the test data. The YOLO v3 model was noted to perform better compared to YOLO v2 achieving a mean average accuracy of 81.86% and loss function as low as 0.3

5.2 Limitations of the study

Training of the YOLO v2 and YOLO v3 model was done on the google colab environment which provides the minimum GPU platform requirements. This has made the training process slower compared to the published articles on YOLO.

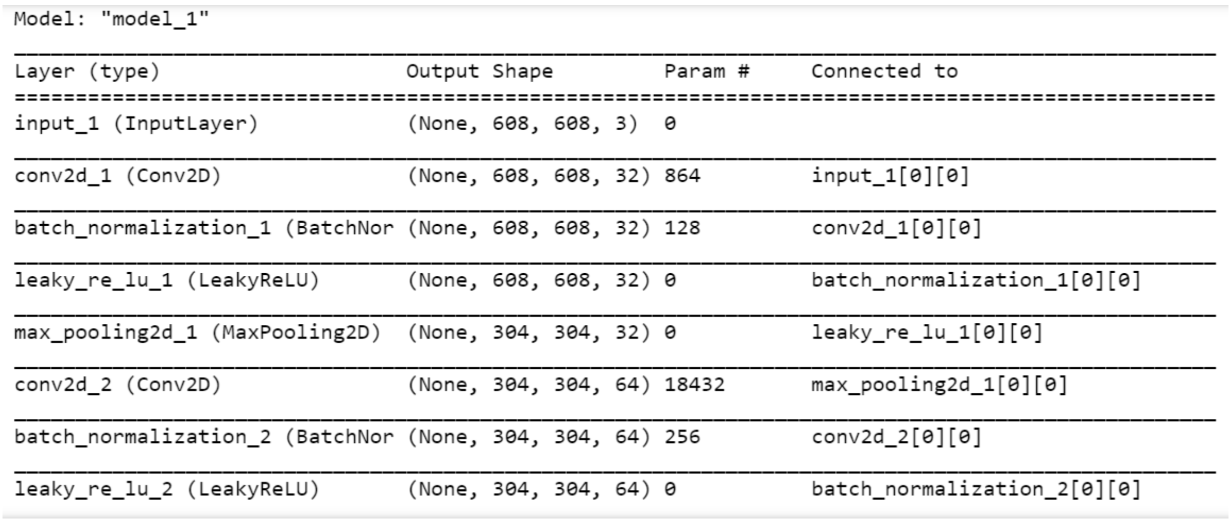
5.3 Recommendations for future work

One key area that can be explored for future work is to enhance the model to detect and count objects for additional product classes in a retail store. Currently the model has been trained using YOLOv2 and YOLO v3 pre- trained weights and further trained with retail store images of bottled soda.

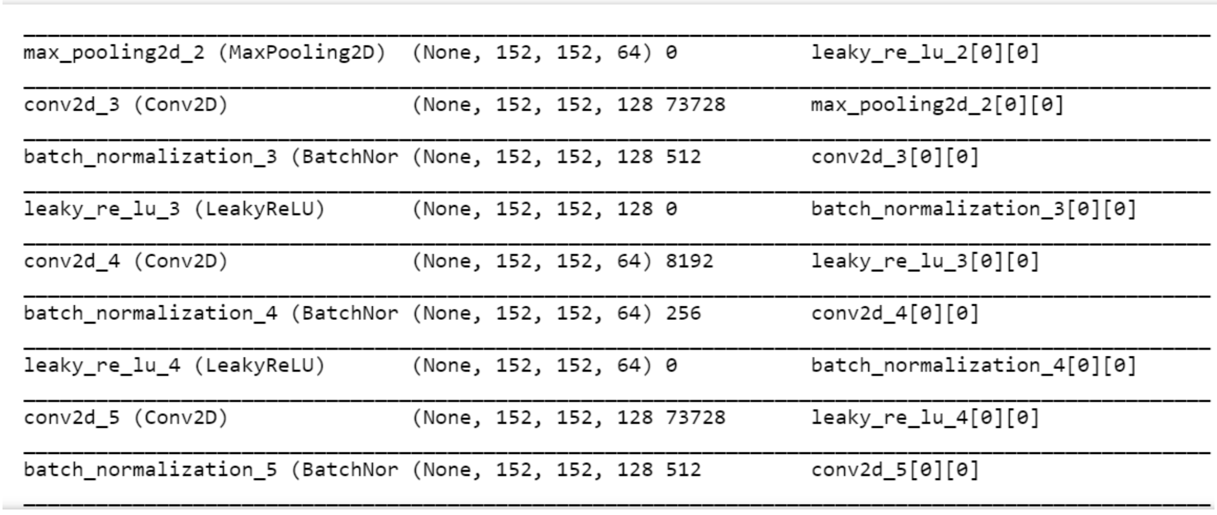
In addition, deployment of this solution in a retail store will require further work to incorporate small shelve cameras that are able to take periodic aerial view images capturing all the items in one view to provide 100% count of the products per shelve.

APPENDIX

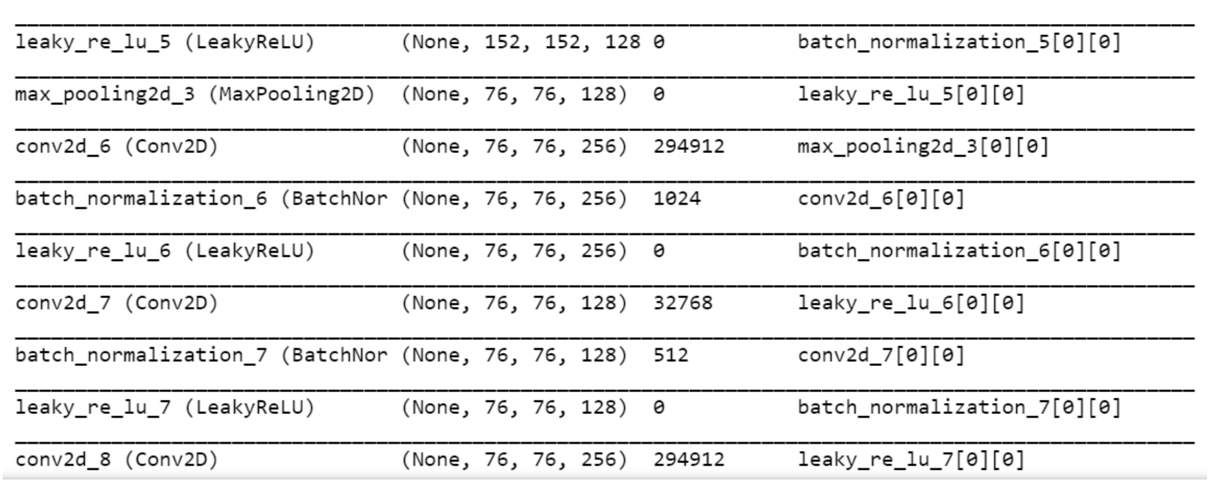
6.1. Architecture of the YOLO model



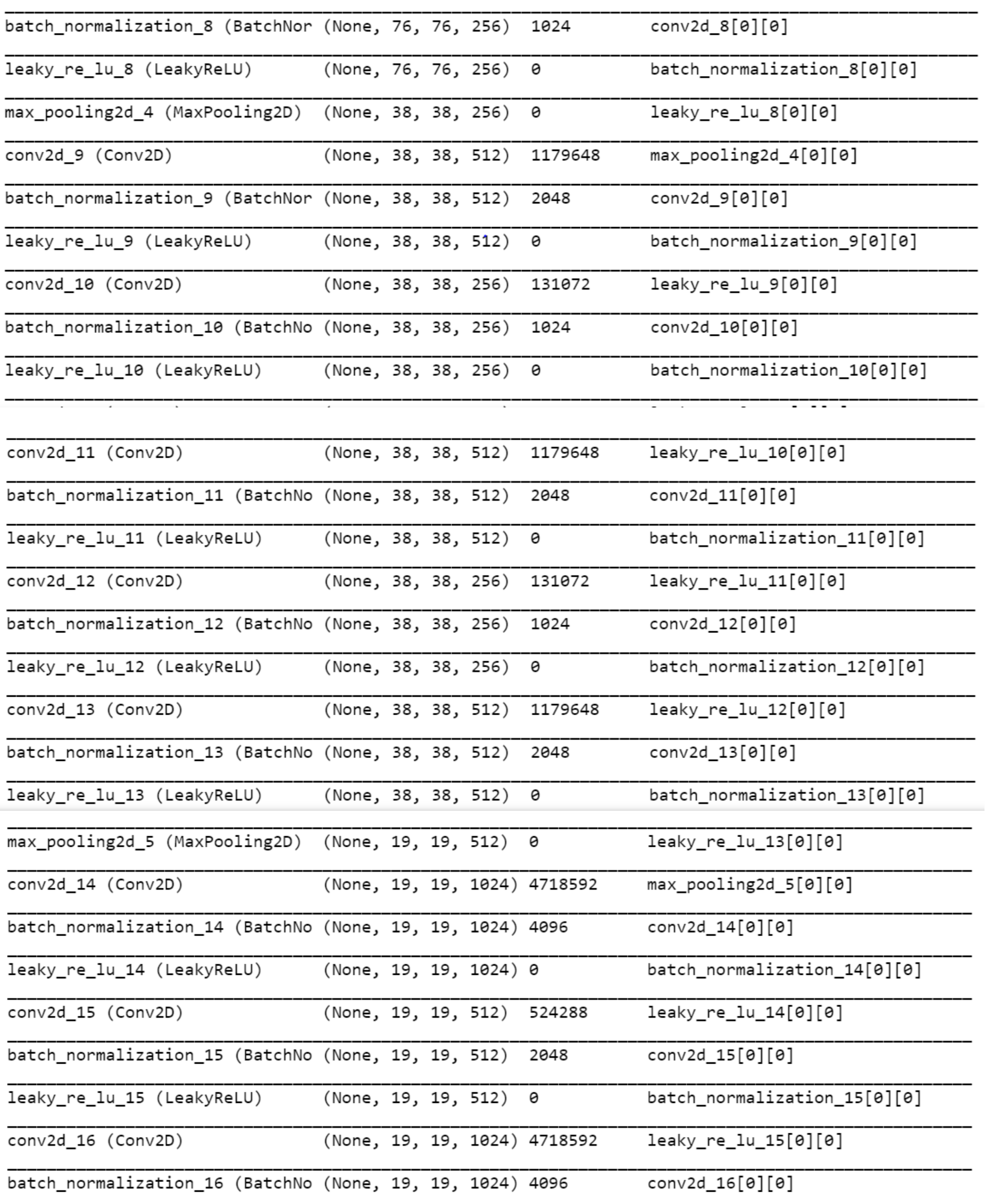
```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 608, 608, 3)  0
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 608, 608, 32) 864         input_1[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 608, 608, 32) 128         conv2d_1[0][0]
__________________________________________________________________________________________________
leaky_re_lu_1 (LeakyReLU)       (None, 608, 608, 32) 0           batch_normalization_1[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 304, 304, 32) 0           leaky_re_lu_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 304, 304, 64) 18432       max_pooling2d_1[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 304, 304, 64) 256         conv2d_2[0][0]
__________________________________________________________________________________________________
leaky_re_lu_2 (LeakyReLU)       (None, 304, 304, 64) 0           batch_normalization_2[0][0]
```



```
__________________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 152, 152, 64) 0           leaky_re_lu_2[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 152, 152, 128 73728       max_pooling2d_2[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 152, 152, 128 512         conv2d_3[0][0]
__________________________________________________________________________________________________
leaky_re_lu_3 (LeakyReLU)       (None, 152, 152, 128 0           batch_normalization_3[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 152, 152, 64) 8192        leaky_re_lu_3[0][0]
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 152, 152, 64) 256         conv2d_4[0][0]
__________________________________________________________________________________________________
leaky_re_lu_4 (LeakyReLU)       (None, 152, 152, 64) 0           batch_normalization_4[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 152, 152, 128 73728       leaky_re_lu_4[0][0]
__________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 152, 152, 128 512         conv2d_5[0][0]
__________________________________________________________________________________________________
```

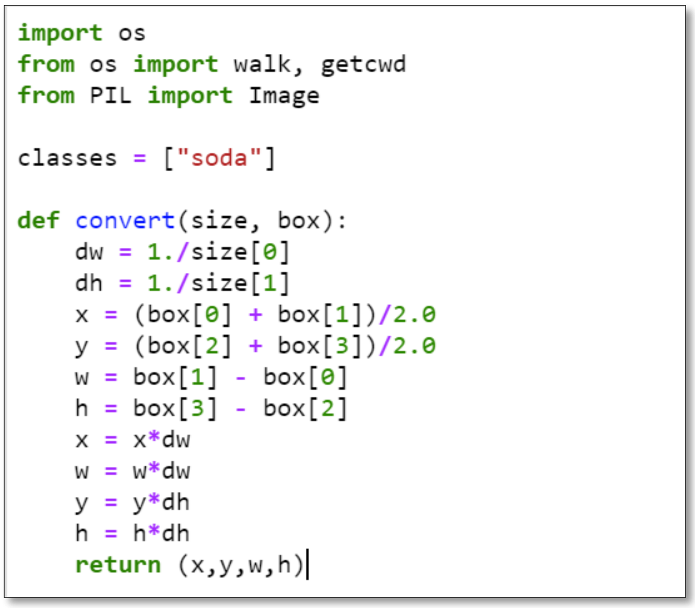


```
__________________________________________________________________________________________________
leaky_re_lu_5 (LeakyReLU)       (None, 152, 152, 128 0           batch_normalization_5[0][0]
__________________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, 76, 76, 128)  0           leaky_re_lu_5[0][0]
__________________________________________________________________________________________________
conv2d_6 (Conv2D)               (None, 76, 76, 256)  294912      max_pooling2d_3[0][0]
__________________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 76, 76, 256)  1024        conv2d_6[0][0]
__________________________________________________________________________________________________
leaky_re_lu_6 (LeakyReLU)       (None, 76, 76, 256)  0           batch_normalization_6[0][0]
__________________________________________________________________________________________________
conv2d_7 (Conv2D)               (None, 76, 76, 128)  32768       leaky_re_lu_6[0][0]
__________________________________________________________________________________________________
batch_normalization_7 (BatchNor (None, 76, 76, 128)  512         conv2d_7[0][0]
__________________________________________________________________________________________________
leaky_re_lu_7 (LeakyReLU)       (None, 76, 76, 128)  0           batch_normalization_7[0][0]
__________________________________________________________________________________________________
conv2d_8 (Conv2D)               (None, 76, 76, 256)  294912      leaky_re_lu_7[0][0]
```



```
________________________________________________________________________________________________
batch_normalization_8 (BatchNor (None, 76, 76, 256)  1024        conv2d_8[0][0]
________________________________________________________________________________________________
leaky_re_lu_8 (LeakyReLU)       (None, 76, 76, 256)  0           batch_normalization_8[0][0]
________________________________________________________________________________________________
max_pooling2d_4 (MaxPooling2D)  (None, 38, 38, 256)  0           leaky_re_lu_8[0][0]
________________________________________________________________________________________________
conv2d_9 (Conv2D)               (None, 38, 38, 512)  1179648     max_pooling2d_4[0][0]
________________________________________________________________________________________________
batch_normalization_9 (BatchNor (None, 38, 38, 512)  2048        conv2d_9[0][0]
________________________________________________________________________________________________
leaky_re_lu_9 (LeakyReLU)       (None, 38, 38, 512)  0           batch_normalization_9[0][0]
________________________________________________________________________________________________
conv2d_10 (Conv2D)              (None, 38, 38, 256)  131072      leaky_re_lu_9[0][0]
________________________________________________________________________________________________
batch_normalization_10 (BatchNo (None, 38, 38, 256)  1024        conv2d_10[0][0]
________________________________________________________________________________________________
leaky_re_lu_10 (LeakyReLU)      (None, 38, 38, 256)  0           batch_normalization_10[0][0]
________________________________________________________________________________________________

________________________________________________________________________________________________
conv2d_11 (Conv2D)              (None, 38, 38, 512)  1179648     leaky_re_lu_10[0][0]
________________________________________________________________________________________________
batch_normalization_11 (BatchNo (None, 38, 38, 512)  2048        conv2d_11[0][0]
________________________________________________________________________________________________
leaky_re_lu_11 (LeakyReLU)      (None, 38, 38, 512)  0           batch_normalization_11[0][0]
________________________________________________________________________________________________
conv2d_12 (Conv2D)              (None, 38, 38, 256)  131072      leaky_re_lu_11[0][0]
________________________________________________________________________________________________
batch_normalization_12 (BatchNo (None, 38, 38, 256)  1024        conv2d_12[0][0]
________________________________________________________________________________________________
leaky_re_lu_12 (LeakyReLU)      (None, 38, 38, 256)  0           batch_normalization_12[0][0]
________________________________________________________________________________________________
conv2d_13 (Conv2D)              (None, 38, 38, 512)  1179648     leaky_re_lu_12[0][0]
________________________________________________________________________________________________
batch_normalization_13 (BatchNo (None, 38, 38, 512)  2048        conv2d_13[0][0]
________________________________________________________________________________________________
leaky_re_lu_13 (LeakyReLU)      (None, 38, 38, 512)  0           batch_normalization_13[0][0]
________________________________________________________________________________________________
max_pooling2d_5 (MaxPooling2D)  (None, 19, 19, 512)  0           leaky_re_lu_13[0][0]
________________________________________________________________________________________________
conv2d_14 (Conv2D)              (None, 19, 19, 1024) 4718592     max_pooling2d_5[0][0]
________________________________________________________________________________________________
batch_normalization_14 (BatchNo (None, 19, 19, 1024) 4096        conv2d_14[0][0]
________________________________________________________________________________________________
leaky_re_lu_14 (LeakyReLU)      (None, 19, 19, 1024) 0           batch_normalization_14[0][0]
________________________________________________________________________________________________
conv2d_15 (Conv2D)              (None, 19, 19, 512)  524288      leaky_re_lu_14[0][0]
________________________________________________________________________________________________
batch_normalization_15 (BatchNo (None, 19, 19, 512)  2048        conv2d_15[0][0]
________________________________________________________________________________________________
leaky_re_lu_15 (LeakyReLU)      (None, 19, 19, 512)  0           batch_normalization_15[0][0]
________________________________________________________________________________________________
conv2d_16 (Conv2D)              (None, 19, 19, 1024) 4718592     leaky_re_lu_15[0][0]
________________________________________________________________________________________________
batch_normalization_16 (BatchNo (None, 19, 19, 1024) 4096        conv2d_16[0][0]
____________________________________________________________________________________________________
leaky_re_lu_16 (LeakyReLU)       (None, 19, 19, 1024) 0           batch_normalization_16[0][0]
____________________________________________________________________________________________________
conv2d_17 (Conv2D)               (None, 19, 19, 512)  524288      leaky_re_lu_16[0][0]
____________________________________________________________________________________________________
batch_normalization_17 (BatchNo (None, 19, 19, 512)  2048        conv2d_17[0][0]
____________________________________________________________________________________________________
leaky_re_lu_17 (LeakyReLU)       (None, 19, 19, 512)  0           batch_normalization_17[0][0]
____________________________________________________________________________________________________
conv2d_18 (Conv2D)               (None, 19, 19, 1024) 4718592     leaky_re_lu_17[0][0]
____________________________________________________________________________________________________
batch_normalization_18 (BatchNo (None, 19, 19, 1024) 4096        conv2d_18[0][0]
____________________________________________________________________________________________________
leaky_re_lu_18 (LeakyReLU)       (None, 19, 19, 1024) 0           batch_normalization_18[0][0]
____________________________________________________________________________________________________
conv2d_19 (Conv2D)               (None, 19, 19, 1024) 9437184     leaky_re_lu_18[0][0]

____________________________________________________________________________________________________
batch_normalization_19 (BatchNo (None, 19, 19, 1024) 4096        conv2d_19[0][0]
____________________________________________________________________________________________________
conv2d_21 (Conv2D)               (None, 38, 38, 64)   32768       leaky_re_lu_13[0][0]
____________________________________________________________________________________________________
leaky_re_lu_19 (LeakyReLU)       (None, 19, 19, 1024) 0           batch_normalization_19[0][0]
____________________________________________________________________________________________________
batch_normalization_21 (BatchNo (None, 38, 38, 64)   256         conv2d_21[0][0]
____________________________________________________________________________________________________
conv2d_20 (Conv2D)               (None, 19, 19, 1024) 9437184     leaky_re_lu_19[0][0]
____________________________________________________________________________________________________
leaky_re_lu_21 (LeakyReLU)       (None, 38, 38, 64)   0           batch_normalization_21[0][0]
____________________________________________________________________________________________________
batch_normalization_20 (BatchNo (None, 19, 19, 1024) 4096        conv2d_20[0][0]
____________________________________________________________________________________________________
space_to_depth_x2 (Lambda)       (None, 19, 19, 256)  0           leaky_re_lu_21[0][0]
____________________________________________________________________________________________________
leaky_re_lu_20 (LeakyReLU)       (None, 19, 19, 1024) 0           batch_normalization_20[0][0]

____________________________________________________________________________________________________
concatenate_1 (Concatenate)      (None, 19, 19, 1280) 0           space_to_depth_x2[0][0]
                                                                  leaky_re_lu_20[0][0]
____________________________________________________________________________________________________
conv2d_22 (Conv2D)               (None, 19, 19, 1024) 11796480    concatenate_1[0][0]
____________________________________________________________________________________________________
batch_normalization_22 (BatchNo (None, 19, 19, 1024) 4096        conv2d_22[0][0]
____________________________________________________________________________________________________
leaky_re_lu_22 (LeakyReLU)       (None, 19, 19, 1024) 0           batch_normalization_22[0][0]
____________________________________________________________________________________________________
conv2d_23 (Conv2D)               (None, 19, 19, 425)  435625      leaky_re_lu_22[0][0]
====================================================================================================
Total params: 50,983,561
```

6.2. Python code to convert the bounding box dimensions to YOLO format.



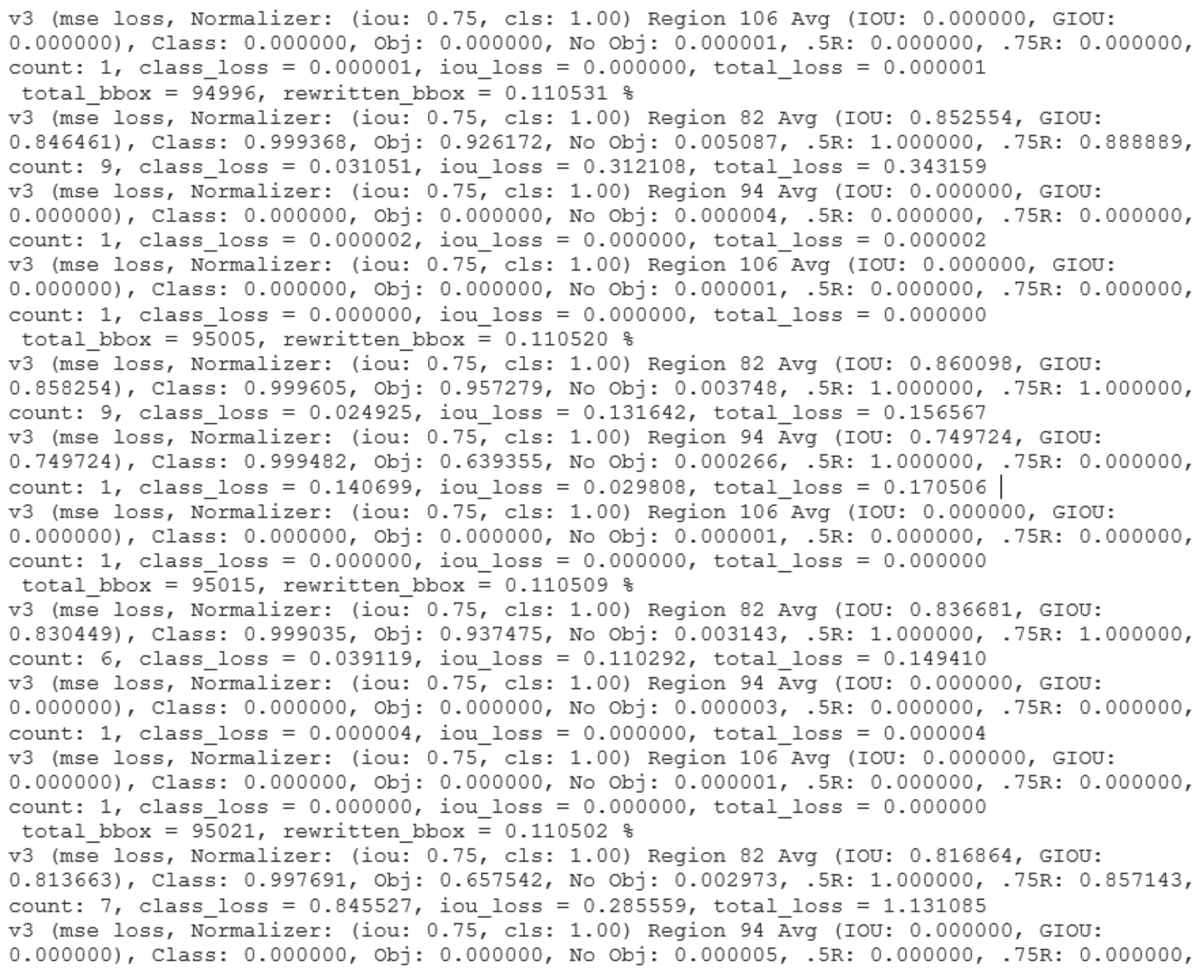
```python
import os
from os import walk, getcwd
from PIL import Image

classes = ["soda"]

def convert(size, box):
    dw = 1./size[0]
    dh = 1./size[1]
    x = (box[0] + box[1])/2.0
    y = (box[2] + box[3])/2.0
    w = box[1] - box[0]
    h = box[3] - box[2]
    x = x*dw
    w = w*dw
    y = y*dh
    h = h*dh
    return (x,y,w,h)
```

Figure 21: Conversion of bounding box dimensions to YOLO format

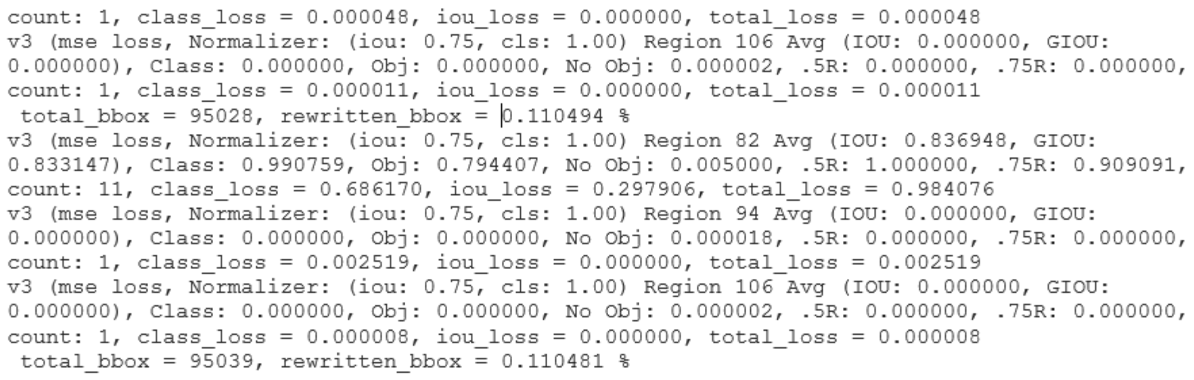
6.3. Sample training output

```
(next mAP calculation at 2000 iterations)
 Last accuracy mAP@0.5 = 79.06 %, best = 80.43 %
 1999: 0.259246, 0.294955 avg loss, 0.000010 rate, 9.552126 seconds, 127936 images, 0.190765
hours left
Loaded: 0.000053 seconds
 Very small path to the image:
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 82 Avg (IOU: 0.802366, GIOU:
0.794813), Class: 0.996088, Obj: 0.607084, No Obj: 0.003271, .5R: 1.000000, .75R: 1.000000,
count: 6, class_loss = 0.761840, iou_loss = 0.214073, total_loss = 0.975913
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 94 Avg (IOU: 0.814794, GIOU:
0.808112), Class: 0.998378, Obj: 0.782958, No Obj: 0.004526, .5R: 1.000000, .75R: 0.807692,
count: 52, class_loss = 2.449728, iou_loss = 2.138028, total_loss = 4.587756
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: 0.538514, GIOU:
0.480532), Class: 0.799927, Obj: 0.006254, No Obj: 0.000011, .5R: 1.000000, .75R: 0.000000,
count: 1, class_loss = 0.258386, iou_loss = 0.154127, total_loss = 0.412513
 total_bbox = 94869, rewritten_bbox = 0.110679 %
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 82 Avg (IOU: 0.858880, GIOU:
0.856015), Class: 0.998362, Obj: 0.754584, No Obj: 0.004185, .5R: 1.000000, .75R: 1.000000,
count: 10, class_loss = 0.420553, iou_loss = 0.173006, total_loss = 0.593558
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 94 Avg (IOU: 0.697203, GIOU:
0.678338), Class: 0.998116, Obj: 0.109732, No Obj: 0.000156, .5R: 1.000000, .75R: 0.500000,
count: 2, class_loss = 0.429167, iou_loss = 0.221631, total_loss = 0.650798
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000002, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.000004, iou_loss = 0.000000, total_loss = 0.000004
 total_bbox = 94881, rewritten_bbox = 0.110665 %
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 82 Avg (IOU: 0.847234, GIOU:
0.845581), Class: 0.998268, Obj: 0.814675, No Obj: 0.002883, .5R: 1.000000, .75R: 0.666667,
count: 6, class_loss = 0.211341, iou_loss = 0.241117, total_loss = 0.452458
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 94 Avg (IOU: 0.777078, GIOU:
0.776644), Class: 0.999491, Obj: 0.637867, No Obj: 0.000056, .5R: 1.000000, .75R: 1.000000,
count: 1, class_loss = 0.042495, iou_loss = 0.084221, total_loss = 0.126716
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000002, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.000033, iou_loss = 0.000000, total_loss = 0.000033
 total_bbox = 94888, rewritten_bbox = 0.110657 %
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 82 Avg (IOU: 0.794026, GIOU:
0.792726), Class: 0.999690, Obj: 0.915238, No Obj: 0.002006, .5R: 1.000000, .75R: 0.750000,
count: 4, class_loss = 0.024071, iou_loss = 0.108710, total_loss = 0.132781
```

```
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 94 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000045, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000001, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000
 total_bbox = 94892, rewritten_bbox = 0.110652 %
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 82 Avg (IOU: 0.893122, GIOU:
0.891621), Class: 0.999450, Obj: 0.964824, No Obj: 0.004179, .5R: 1.000000, .75R: 1.000000,
count: 8, class_loss = 0.008884, iou_loss = 0.117922, total_loss = 0.126806
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 94 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000003, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000001, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000
 total_bbox = 94900, rewritten_bbox = 0.110643 %
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 82 Avg (IOU: 0.925744, GIOU:
0.925423), Class: 0.999962, Obj: 0.981857, No Obj: 0.001732, .5R: 1.000000, .75R: 1.000000,
count: 3, class_loss = 0.000572, iou_loss = 0.057187, total_loss = 0.057760
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 94 Avg (IOU: 0.820545, GIOU:
0.812799), Class: 0.999572, Obj: 0.886118, No Obj: 0.001743, .5R: 1.000000, .75R: 0.875000,
count: 16, class_loss = 0.298406, iou_loss = 0.629953, total_loss = 0.928359
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: 0.157612, GIOU:
0.055748), Class: 0.624516, Obj: 0.001624, No Obj: 0.000005, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.285048, iou_loss = 0.185713, total_loss = 0.470762
 total_bbox = 94920, rewritten_bbox = 0.110619 %
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 82 Avg (IOU: 0.894711, GIOU:
0.893783), Class: 0.999769, Obj: 0.940274, No Obj: 0.003776, .5R: 1.000000, .75R: 1.000000,
count: 9, class_loss = 0.054667, iou_loss = 0.085429, total_loss = 0.140096
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 94 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000038, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.000140, iou_loss = 0.000000, total_loss = 0.000140
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000001, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.000002, iou_loss = 0.000000, total_loss = 0.000002
 total_bbox = 94929, rewritten_bbox = 0.110609 %
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 82 Avg (IOU: 0.884494, GIOU:
0.882610), Class: 0.999674, Obj: 0.936406, No Obj: 0.002632, .5R: 1.000000, .75R: 1.000000,
count: 6, class_loss = 0.080155, iou_loss = 0.080643, total_loss = 0.160798
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 94 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000036, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000001, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000
 total_bbox = 94935, rewritten_bbox = 0.110602 %
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 82 Avg (IOU: 0.838335, GIOU:
0.837297), Class: 0.999645, Obj: 0.931083, No Obj: 0.002556, .5R: 1.000000, .75R: 1.000000,
count: 6, class_loss = 0.185011, iou_loss = 0.104110, total_loss = 0.289122
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 94 Avg (IOU: 0.816270, GIOU:
0.812109), Class: 0.998659, Obj: 0.683569, No Obj: 0.003362, .5R: 1.000000, .75R: 0.787879,
count: 33, class_loss = 2.519956, iou_loss = 1.695547, total_loss = 4.215503
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000006, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.001744, iou_loss = 0.000000, total_loss = 0.001744
 total_bbox = 94974, rewritten_bbox = 0.110557 %
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 82 Avg (IOU: 0.843228, GIOU:
0.842736), Class: 0.998634, Obj: 0.829905, No Obj: 0.004434, .5R: 1.000000, .75R: 0.900000,
count: 10, class_loss = 0.640320, iou_loss = 0.165642, total_loss = 0.805962
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 94 Avg (IOU: 0.875179, GIOU:
0.875179), Class: 0.999549, Obj: 0.670447, No Obj: 0.000150, .5R: 1.000000, .75R: 1.000000,
count: 1, class_loss = 0.098618, iou_loss = 0.043893, total_loss = 0.142511
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000001, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000
 total_bbox = 94985, rewritten_bbox = 0.110544 %
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 82 Avg (IOU: 0.854327, GIOU:
0.851286), Class: 0.993362, Obj: 0.811730, No Obj: 0.003676, .5R: 1.000000, .75R: 0.875000,
count: 8, class_loss = 0.277713, iou_loss = 0.228623, total_loss = 0.506335
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 94 Avg (IOU: 0.847242, GIOU:
0.841787), Class: 0.998767, Obj: 0.303361, No Obj: 0.000218, .5R: 1.000000, .75R: 1.000000,
count: 3, class_loss = 0.510345, iou_loss = 0.182462, total_loss = 0.692807
```

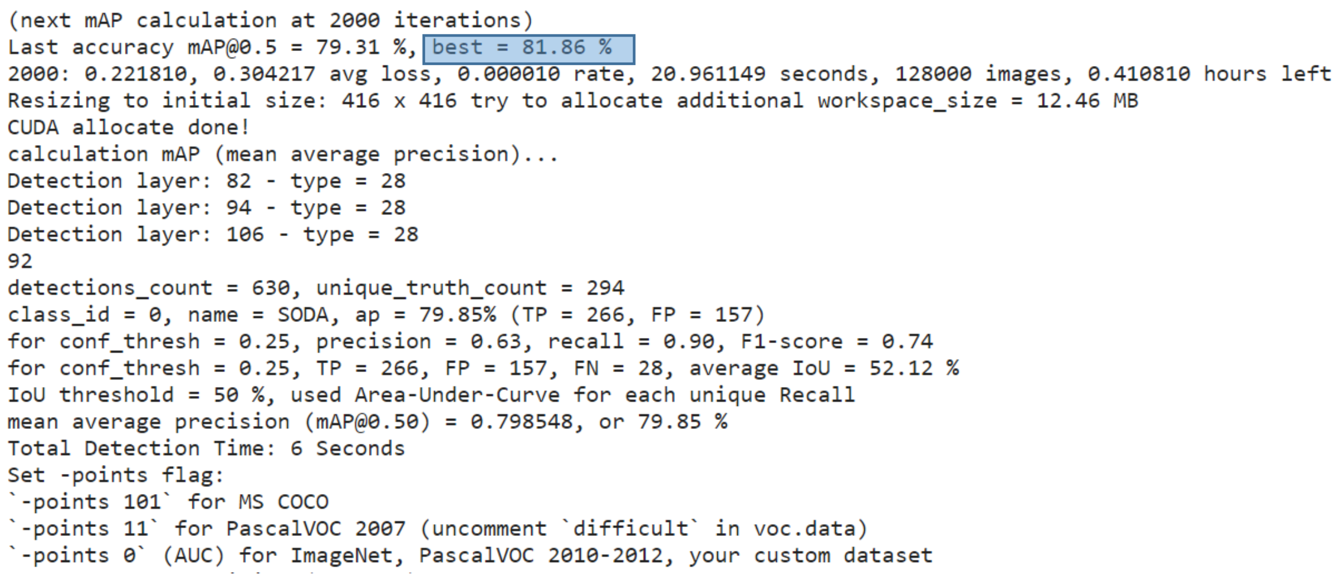


```
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000001, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.000001, iou_loss = 0.000000, total_loss = 0.000001
 total_bbox = 94996, rewritten_bbox = 0.110531 %
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 82 Avg (IOU: 0.852554, GIOU:
0.846461), Class: 0.999368, Obj: 0.926172, No Obj: 0.005087, .5R: 1.000000, .75R: 0.888889,
count: 9, class_loss = 0.031051, iou_loss = 0.312108, total_loss = 0.343159
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 94 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000004, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.000002, iou_loss = 0.000000, total_loss = 0.000002
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000001, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000
 total_bbox = 95005, rewritten_bbox = 0.110520 %
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 82 Avg (IOU: 0.860098, GIOU:
0.858254), Class: 0.999605, Obj: 0.957279, No Obj: 0.003748, .5R: 1.000000, .75R: 1.000000,
count: 9, class_loss = 0.024925, iou_loss = 0.131642, total_loss = 0.156567
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 94 Avg (IOU: 0.749724, GIOU:
0.749724), Class: 0.999482, Obj: 0.639355, No Obj: 0.000266, .5R: 1.000000, .75R: 0.000000,
count: 1, class_loss = 0.140699, iou_loss = 0.029808, total_loss = 0.170506
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000001, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000
 total_bbox = 95015, rewritten_bbox = 0.110509 %
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 82 Avg (IOU: 0.836681, GIOU:
0.830449), Class: 0.999035, Obj: 0.937475, No Obj: 0.003143, .5R: 1.000000, .75R: 1.000000,
count: 6, class_loss = 0.039119, iou_loss = 0.110292, total_loss = 0.149410
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 94 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000003, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.000004, iou_loss = 0.000000, total_loss = 0.000004
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000001, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss = 0.000000
 total_bbox = 95021, rewritten_bbox = 0.110502 %
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 82 Avg (IOU: 0.816864, GIOU:
0.813663), Class: 0.997691, Obj: 0.657542, No Obj: 0.002973, .5R: 1.000000, .75R: 0.857143,
count: 7, class_loss = 0.845527, iou_loss = 0.285559, total_loss = 1.131085
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 94 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000005, .5R: 0.000000, .75R: 0.000000,
```



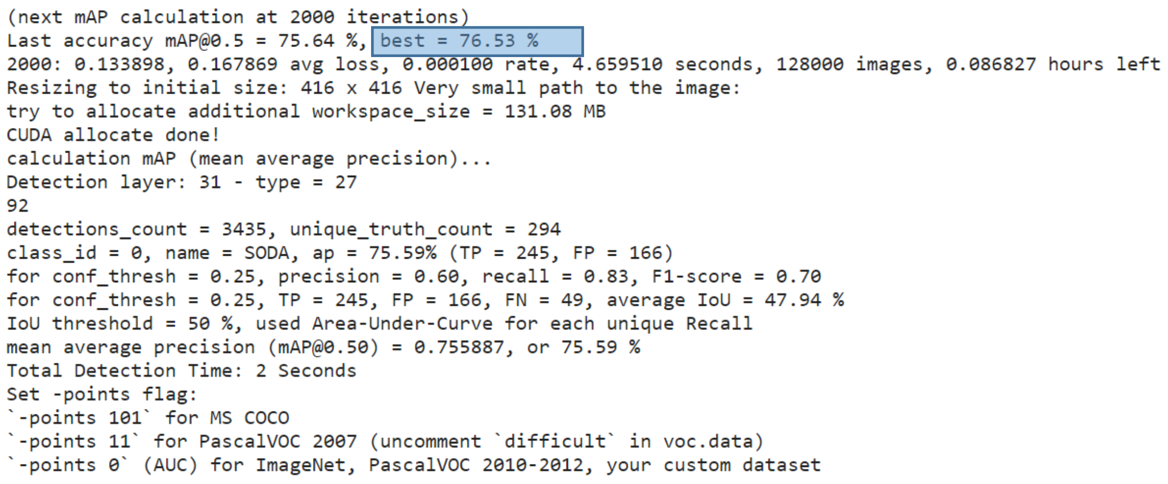
```
count: 1, class_loss = 0.000048, iou_loss = 0.000000, total_loss = 0.000048
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000002, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.000011, iou_loss = 0.000000, total_loss = 0.000011
 total_bbox = 95028, rewritten_bbox = 0.110494 %
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 82 Avg (IOU: 0.836948, GIOU:
0.833147), Class: 0.990759, Obj: 0.794407, No Obj: 0.005000, .5R: 1.000000, .75R: 0.909091,
count: 11, class_loss = 0.686170, iou_loss = 0.297906, total_loss = 0.984076
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 94 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000018, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.002519, iou_loss = 0.000000, total_loss = 0.002519
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: 0.000000, GIOU:
0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.000002, .5R: 0.000000, .75R: 0.000000,
count: 1, class_loss = 0.000008, iou_loss = 0.000000, total_loss = 0.000008
 total_bbox = 95039, rewritten_bbox = 0.110481 %
```

6.4. YOLO v3 model accuracy evaluation output



```
(next mAP calculation at 2000 iterations)
Last accuracy mAP@0.5 = 79.31 %, best = 81.86 %
2000: 0.221810, 0.304217 avg loss, 0.000010 rate, 20.961149 seconds, 128000 images, 0.410810 hours left
Resizing to initial size: 416 x 416 try to allocate additional workspace_size = 12.46 MB
CUDA allocate done!
calculation mAP (mean average precision)...
Detection layer: 82 - type = 28
Detection layer: 94 - type = 28
Detection layer: 106 - type = 28
92
detections_count = 630, unique_truth_count = 294
class_id = 0, name = SODA, ap = 79.85% (TP = 266, FP = 157)
for conf_thresh = 0.25, precision = 0.63, recall = 0.90, F1-score = 0.74
for conf_thresh = 0.25, TP = 266, FP = 157, FN = 28, average IoU = 52.12 %
IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.798548, or 79.85 %
Total Detection Time: 6 Seconds
Set -points flag:
`-points 101` for MS COCO
`-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
`-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset
```

6.5. YOLO v2 model accuracy evaluation output



```
(next mAP calculation at 2000 iterations)
Last accuracy mAP@0.5 = 75.64 %, best = 76.53 %
2000: 0.133898, 0.167869 avg loss, 0.000100 rate, 4.659510 seconds, 128000 images, 0.086827 hours left
Resizing to initial size: 416 x 416 Very small path to the image:
try to allocate additional workspace_size = 131.08 MB
CUDA allocate done!
calculation mAP (mean average precision)...
Detection layer: 31 - type = 27
92
detections_count = 3435, unique_truth_count = 294
class_id = 0, name = SODA, ap = 75.59% (TP = 245, FP = 166)
for conf_thresh = 0.25, precision = 0.60, recall = 0.83, F1-score = 0.70
for conf_thresh = 0.25, TP = 245, FP = 166, FN = 49, average IoU = 47.94 %
IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.755887, or 75.59 %
Total Detection Time: 2 Seconds
Set -points flag:
`-points 101` for MS COCO
`-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
`-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset
```

REFERENCES

Barbedo, J. G. A., 2012. A review on methods for automatic counting of objects in digital images. *IEEE,* 10(5), pp. 2112-2124.

Barratt, M., Rabinovich, E. & 2010., A. S., 2010. Inventory Accuracy: Essential, but Often Overlooked. *Supply Chain Management Review,* pp. 36-46.

Bhausaheb Shivajirao Shinde, A. D., 2011. The Origins of Digital Image Processing & Application areas in Digital Image Processing Medical Images. *IOSR Journal of Engineering,* 1(1), pp. 066-071.

Canziani, A., Paszke, A. & Culurciello, E., 2016. An Analysis of Deep Neural Network Models for Practical Applications. *ArXiv e-prints.*

D. Hoiem, Y. C. D., 2012. Diagnosing error in object detectors. *Computer Vision–ECCV,* pp. 340 - 353.

Everingham, M. et al., 2015. The pascal visual object classes challenge: A retrospective. *Internalitonal journal of computer vision,* 111(1), pp. 98-136.

Felzenszwalb, P. F., Girshick, R. B., McAllester, D. & Ramanan., D., 2010. Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and,* 32(9), pp. 1627-1649.

French, M. G., Fisher, D. M. H., Mackiewicz, D. M. & Needle, D. C. L., 2015. *Convolutional Neural Networks for Counting fish in fisheries Surveillance Video.* Swansea, UK, Workshop on Machine Vision of Animals and their Behaviour,.

Goodfellow, I. J. et al., 2014. *Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks,* Mountain View, CA: Google Inc..

Guo, Y. et al., 2015. Deep learning for visual understanding: A review. *ELSEVIER,* Volume 187, pp. 27-48.

He, K., Zhang, X., Ren, S. & Sun, J., 2016. *Deep residual learning for image recognition.* s.l., Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.

Ishfaq, R. & Raja, U., 2019. Effectiveness of frequent inventory audits in retail stores: an emprical evaluation. *International journal of logistics management ISSN: 0957-4093,* 30(4).

Kamali, A., 2018. Physical Inventory Counting. *CiiT International Journal of Software Engineering And Technology,* 10(10), pp. 197-202.

Kull, T. J., Sodero, A., Barratt, M. & Rabinovich, E., 2013. Investigating the effect of daily inventory record inaccuracy in multichannel retailing. *Journal of Business logistics .*

Lee, H. & Özer., Ö., 2007. Unlocking the Value of Rfid. *Production and Operations Management,* 16(1), pp. 40 - 64.

Lin, M., Q., C. & S. Yan, 2013. Network in network. *ArXiv e-prints.*

Luoh, L., Huang, C.-C. & Liu, H.-Y., 2010. Image processing based emotion recognition. *2010 International Conference on System Science and Engineering,* pp. 491-494.

P. F. Felzenszwalb, R. B. G. D. M. a. D. R., 2010. Object detection with discriminatively trained part. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 32(9), pp. 1627 - 1645.

Rahnemoonfar, M. & Sheppard, C., 2017. Deep Count: Fruit Counting Based on Deep Simulated Learning. *Multidisciplinary Digital Publishing Institute,* Volume 174.

Redmon, J., Divvala, S., Girshick, R. & Farhadi, A., 2016. You only look once unified real time object detection. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Simonyan, K. & Zisserman, A., 2015. *VERY DEEP CONVOLUTIONAL NETWORKS for large scale image recognition.* s.l., International Conference on Learning Representations.

Soltania, M. M. et al., 2017. *Evaluating the Performance of Convolutional Neural Network for Classifying Equipment on Construction Sites.* Taipei, Taiwan, The International Association for Automation and Robotics in Construction, pp. 509-516.

Szegedy, C. et al., 2015. *Going deeper with convolutions.* s.l., Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.

Szegedy, C. et al., 2016. *Rethinking the inception architecture for computer vision.* s.l., Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.

Wang, W.-J., Chang, J.-W., Haung, S.-F. & Wang, R.-J., 2016. Human Posture Recognition Based on Images Captured by the Kinect Sensor. *International Journal of Advanced Robotic Systems,* 13(2).