

https://dspace.org/

Automatic vision-based firearm recognition using deep learning approaches: a comparative analysis

Turundu, Safiya

2021
https://hdl.handle.net/10539/33100

Downloaded from a DSpace repository. https://dspace.org/

Automatic Vision-based Firearm Recognition using Deep Learning Approaches: A Comparative Analysis

Safiya Turundu (323438)
Supervised by Prof. Turgay Celik



UNIVERSITY OF THE
WITWATERSRAND,
JOHANNESBURG

University of the Witwatersrand
School of Computer Science and Applied Mathematics (CSAM)

A dissertation submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, in partial fulfilment of the requirements for the degree of Master of Science

September, 2021

DECLARATION

I declare that this dissertation is my own, unaided work. It is being submitted for the Degree of Master of Science at the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination at any other University.



(Signature of candidate)

On the **7th** day of **September** 20**21** in **Johannesburg**

ACKNOWLEDGEMENTS

I would firstly like to thank my supervisor, Prof. Turgay Celik, for guiding me and teaching me standard research techniques. Secondly, I would like to acknowledge my Mother, Father, and husband for their tremendous support throughout this journey. They never gave up on me. And last but not least, I thank God for giving me the strength to keep going and stay focused.

ABSTRACT

Firearms held up in an environment not designated for legal firearm activity is an indicator of a dangerous situation. This research proposes a vision-based firearm detection model which is able to assist in such a scenario in public spaces such as malls, airports, and residential areas. The aim is to evaluate the performance of various deep learning architectures based on their execution of detecting visible firearms in still images. The approach involves using publicly available datasets for deep learning detectors to learn from by extracting features related to firearms. Detection results are compared with one another based on the F1-score, a function of precision and recall measures for detector evaluation. The deep learning architectures to be evaluated are a mixture of existing state-of-the-art single-stage and two-stage object detectors typically used in deep learning. These include the Single Shot Multi-box Detector (SSD), You Only Look Once (YOLO), Region-based Fully Convolutional Network (RFCN), and Faster Region-Convolutional Neural Network (Faster R-CNN). The results obtained by the detectors describe how accurately each one performs the classification and localisation tasks in comparison with one another. Furthermore, inferencing speed of each detector after being fully trained is also compared.

Contents

List of Figures v

List of Tables vii

List of Abbreviations viii

1 Introduction 1

2 Background and Related Research 6
2.1 Literature review . . . 6
2.1.1 Generic object detection using deep learning . . . 6
2.1.2 Experimental evaluation on generic object detection . . . 7
2.1.3 Object class differentiation for firearm detection use . . . 8
2.1.4 Dataset formation and preprocessing . . . 9
2.1.5 Metrics used to test performance of firearm detection methods . . . 11
2.2 Detection Approaches . . . 11
2.2.1 Traditional Approaches . . . 12
2.2.2 Deep Learning . . . 14
2.2.3 Hybrid Approach . . . 22

3 Single Stage Object Detectors 23
3.1 Overview of single stage object detectors . . . 23
3.2 Single Shot Multibox Detector (SSD) . . . 25
3.2.1 SSD Structure . . . 26
3.2.2 Default Boxes . . . 26
3.2.3 Loss Function . . . 27
3.2.4 Non-Maximum Suppression . . . 28
3.2.5 Deconvolutional Single Shot Detector (DSSD) . . . 29
3.3 You Only Look Once (YOLO) . . . 30
3.3.1 YOLO structure . . . 30
3.3.2 Predictions using bounding boxes . . . 31
3.3.3 YOLO loss function . . . 32
3.4 YOLOv2 (YOLO9000) . . . 33
3.4.1 Error analysis of YOLO . . . 34
3.4.2 Network design . . . 34
3.4.3 Training . . . 34
3.4.4 Predictions using anchor boxes . . . 36
3.5 YOLOv3 . . . 38

3.5.1 Network design . . . 38
3.5.2 Class prediction . . . 38
3.6 Conclusion . . . 40

4 Two-Stage Object Detectors **42**
4.1 Overview of two-stage object detectors . . . 42
4.2 Region Convolutional Neural Networks (R-CNN) . . . 44
4.2.1 Feature extraction . . . 44
4.2.2 Training . . . 44
4.2.3 Bounding box regression . . . 45
4.3 Fast R-CNN . . . 46
4.3.1 Limitations of R-CNN . . . 46
4.3.2 Architectural Design of Fast R-CNN . . . 46
4.3.3 Fine-tuning for training . . . 47
4.3.4 Faster detection . . . 49
4.4 Faster R-CNN . . . 49
4.4.1 Convolutional layers . . . 49
4.4.2 Anchor boxes . . . 50
4.4.3 Region Proposal Network (RPN) . . . 51
4.4.4 RoI pooling and detection . . . 52
4.5 Region-based Fully Convolutional Networks (RFCN) . . . 52
4.5.1 RFCN Architecture . . . 52
4.5.2 Position-sensitive score maps & Position-sensitive RoI . . . 53
4.5.3 Training and Inferencing . . . 54
4.6 Conclusion . . . 55

5 Research Methodology **57**
5.1 Research question . . . 57
5.2 Aim and Objectives . . . 57
5.3 Research method . . . 58
5.3.1 Dataset . . . 58
5.3.2 Preprocessing . . . 59
5.3.3 Deep learning classifiers and training . . . 59
5.3.4 Testing, Evaluation, and Optimisation . . . 60
5.4 Conclusion . . . 61

6 Experimental results and discussions **63**
6.1 Introduction . . . 63
6.1.1 Training software . . . 63
6.2 Analysis of performance per classifier . . . 64
6.2.1 F_1 score evaluation . . . 64
6.2.2 Classification, Localisation, and Total Loss . . . 64
6.2.3 Inference speed evaluation . . . 65
6.2.4 SSD Observations . . . 65
6.2.5 YOLO Observations . . . 69
6.2.6 RFCN Observations . . . 74
6.2.7 Faster R-CNN Observations . . . 78
6.3 Comparison of results . . . 82

6.3.1 Comparison of Single stage object detectors 82
6.3.2 Comparison of Two stage object detectors 84
6.3.3 Classifier Results 85
6.4 Answers to research questions 85
6.5 Conclusion 87

7 Conclusion **88**

References **90**

List of Figures

1.1 Violent crimes - statistics for weapons of choice in South Africa in 2018 [eNCA 2018] . . . 2
1.2 Two AK47 rifles and a 9mm pistol with ammunition found through a tip-off [Writer 2016]. . . . 3

2.1 Comparison of household object to handheld gun *Sources:* (a)[Mecanico] and (b)[Velocidad] . . . 9
2.2 Replica-gun sample with equivalent electromagnetic image [Alqubaa and Tian 2012]. . . . 10
2.3 VGG16 architectural design [Nash *et al.* 2018]. . . . 16
2.4 Residual Learning - shortcut connection [He *et al.* 2016]. . . . 18
2.5 Inception model version 1 (left); Inception model version 2 (right) [Szegedy *et al.* 2016]. . . . 21

3.1 Example of $B_1 \cap B_2$ (left) vs. $B_1 \cup B_2$ (right) . . . 24
3.2 Firearm detection output predicted with red bounding box, and green ground truth box containing a firearm as the target object. . . . 24
3.3 Output tensor structure [Noh *et al.* 2018] . . . 25
3.4 SSD structure with an input image of size $300 \times 300 \times 3$ [Liu *et al.* 2016] . . . 25
3.5 Input image of a cat and dog within their own ground truth box, alongside an 8 x 8 and a 4 x 4 feature map with default boxes predicting the localization and classification output. The blue boxes correspond to the cat, while the red corresponds to the dog [Liu *et al.* 2016] . . . 26
3.6 Example of NMS applied to an image with multiple default bounding boxes [Hosang *et al.* 2017] . . . 28
3.7 Structure of DSSD with additional deconvolutional layers [Fu *et al.* 2017] . . . 29
3.8 Sample input image fed into the YOLO algorithm for detecting a person, dog and cat. The image undergoes a generalised three-step process performed by the network which leads to detecting the target objects [Redmon *et al.* 2016] . . . 30
3.9 YOLO network design [Geethapriya *et al.* 2019] . . . 31
3.10 Subtree for classifying a Dalmatian . . . 36
3.11 Bounding box location predictions [Redmon and Farhadi 2017] . . . 37
3.12 YOLOv3 Network Architecture [Jing *et al.* 2020] . . . 39

4.1 Two-stage object detector architecture. Inputs images are fed into a sequence of convolutional layers (*Conv*) represented by the yellow cubes. Feature maps are then generated and passed into the Region Proposal network. The blue cubes represent convolutional layers in this network from which classification (*cls*) and localisation (*loc*) are predicted.[Jiao *et al.* 2019] . . . 43
4.2 Fast R-CNN Architecture [Girshick 2015] . . . 47
4.3 Faster R-CNN network consisting of convolutional layers and the RPN [Ren *et al.* 2015a] . . . 50
4.4 Architecture of RFCN from end-to-end [Dai *et al.* 2016] . . . 53

5.1 Examples of possible detection outputs . . . 61

6.1 SSD firearm detection samples from Group B classifier results. SSD is easily able to detect firearms that contrast with background colours. The top-right and bottom-left images each contain a firearm which are small in size and are coloured similarly to its background. This creates a blending effect which caused the SSD classifier to predict a FN. . . . 66
6.2 Loss graphs produced during training for SSD . . . 67
6.2 Loss graphs produced during training for SSD . . . 68
6.3 YOLO firearm detection samples taken from Group C results. Similar to SSD visual results, YOLO was mostly able to correctly classify and locate the firearm except where the firearm colour blended into the background as seen in the top-right and bottom-left images. . . . 70
6.4 The YOLO loss graphs describe the change in average loss values over time during the training process. The expected output is a downward sloping graph which starts off with the loss dropping quickly in the first few hundred iterations. The graph begins to average out as the average loss value tends closer to 0. The current average loss is depicted on the y-axis, while the iteration/steps are shown on the x-axis. The blue symbols beneath each graph represent the average loss at the end of training (current avg loss), the last iteration before cut-off (iteration), and time left for training which is optionally pre-defined (approx. time left). . . . 72
6.5 RFCN firearm detection samples taken from results obtained by Group B. In these images, all guns were detected accurately. The confidence score of the top-right and bottom-left images is lower than those scored in other images due to the small size of the firearms. This shows that RFCN is able to better process larger sized firearms than those at a small scale. . . . 75
6.6 RFCN loss graphs . . . 76
6.6 RFCN loss graphs . . . 77
6.7 Faster R-CNN firearm detection samples taken from the results of Group E. Due to the use of RoI pooling and anchor boxes, this classifier has the added advantage of being able to detect firearms at varying scales. As a result, it was able to accurately predict all firearms present in these samples with an accompanying high confidence score of 99%, which is higher than those predicted by RFCN for the smaller firearms. . . . 79
6.7 Loss graphs produced during training for Faster R-CNN . . . 81

List of Tables

1.1 Guns Most Prominent in Southern African Crime [Keegan 2005] 4

2.1 Comparison of models based on mAP, inferencing speed, and FPS measures using the PASCAL VOC 2007 Test set [Zhao *et al.* 2019] 8

3.1 Darknet-19 convolution and maxpooling layer operations [Redmon and Farhadi 2017] 35

4.1 Softmax vs SVM in Fast R-CNN network [Girshick 2015] 48

6.1 Precision, Recall, and F_1 score results for SSD using K-fold validation . . . 66
6.2 Loss values obtained for SSD trained model 67
6.3 Inferencing speed of SSD 69
6.4 Precision, Recall, and F_1 score results for YOLO using K-fold validation . . 70
6.5 YOLO overall loss 71
6.6 Inferencing speed of YOLO 73
6.7 Precision, Recall, and F_1 score results for RFCN using K-fold validation . . 74
6.8 Loss values obtained for RFCN trained model 74
6.9 Inferencing speed of RFCN 77
6.10 Precision, Recall, and F_1 score results for Faster R-CNN using K-fold validation 78
6.11 Loss values obtained for Faster R-CNN trained models 79
6.12 Inferencing speed of Faster R-CNN 81
6.13 Model performance output 85
6.14 Comparison of model inferencing speeds 86
6.15 Comparison of model F_1 scores 87

List of Abbreviations

AATpI Alarm Activation Time per Interval

ANN Artificial Neural Network

BM Batch Normalization

BRIEF Binary Robust Independent Elementary Features

CNN Convolutional Neural Network

COCO common Objects in Context

Conv Convolutional layers

DaCoLT Darkening and Contrast at Learning and Test Stages

EM Electromagnetic

FN False Negative

FP False Positive

FPN Feature Pyramid Network

FREAK Fast Retina Keypoint

GPU Graphical Processing Unit

ILSVRC ImageNet Large Scale Visual Recognition Challenge

IoU Intersection over Union

mAP Mean Average Precision

NMS Non-Maximum Suppression

PASCAL-VOC Pascal Visual Object Classes

R-CNN Region-Convolutional Neural Network

ReLu Rectified Liner Unit

ResNet Residual Network

RFCN Region-based Fully Convolutional Network

RoI Region of Interest

RPN Region Proposal Network

SCI^2 Soft Computing and Intelligent Information Systems

SGBM Semi-Global Block Matching

SGD Stochastic Gradient Descent

SIFT Scale-Invariant Feature Transform

SSD Single Shot Multi-box Detector

SURF Fast Retina Keypoint

SVD Singular Value Decomposition

SVM Support Vector Machine

TN True Negative

TP True Positive

YOLO You Only Look Once

Chapter 1

Introduction

The *Firearms Control Act (Act No. 60 of 2000)* states that the right to possess a firearm is not guaranteed by law (in South Africa), and such a right is granted under limited circumstances [Attorneys 2017]. The debate on whether or not firearms should be legalised for private ownership amongst civilians has been long going and continues to be under the spotlight. Both arguments have their associated pros and cons. According to a weapons watchdog organisation, there are approximately over 5 million guns in the possession of people in South Africa. However, only 3 million out of that number are legally owned and have followed correct procedures for registration [Nagtegaal 2018].

There are certain activities which under the Firearms Control Act have been deemed legal. Such uses include self-defence, occasional and dedicated hunters (sports shooters), and business purposes (security companies). However, firearms (licensed or unlicensed) are being abused for nefarious reasons in places not designated for its use. Therefore, what more can be done using deep learning techniques to detect a firearm in a dangerous situation, where humans fail to do so? This research is aimed at comparing and contrasting deep learning classifiers that are able to detect firearms when trained appropriately. The results will measure the accuracy of classifier recognition using a confidence score and speed of inferencing in seconds at the testing stage on a set of images containing one or more firearms. This will ultimately show which classifiers would be best suited in real-time for firearm detection in dangerous situations where firearms have not been authorised to be used by civilians. Suitable detectors can be installed onto cameras and implemented in public places such as airports, train stations, malls, etc.

An average of 57 people are killed a day in South Africa, according to statistics published by the South Africa Police Service (SAPS) for the 2017/2018 period [Sicetsha 2018]. Crime statistics also show that the murder rate in South Africa has increased by 3.4% as of September 2019, which works out to 58 murders/day [Check 2020]. Police Minister Bheki Cele released a statement based on crime statistics, saying that most violent crimes are committed through the use of firearms. According to national crime statistics, guns are most commonly used by criminals for murders and aggravated robberies. Displayed in Fig. 1.1 are statistics compiled by the SAPS of national weapon usage frequency in 2018.

Recorded crime cases have shown that guns are fast becoming the first choice for criminals to intimidate, threaten, and harm people. Guns have been used in various forms of crimes such as car hijackings, robbing of motorists (colloquially known as “Smash and Grab”),

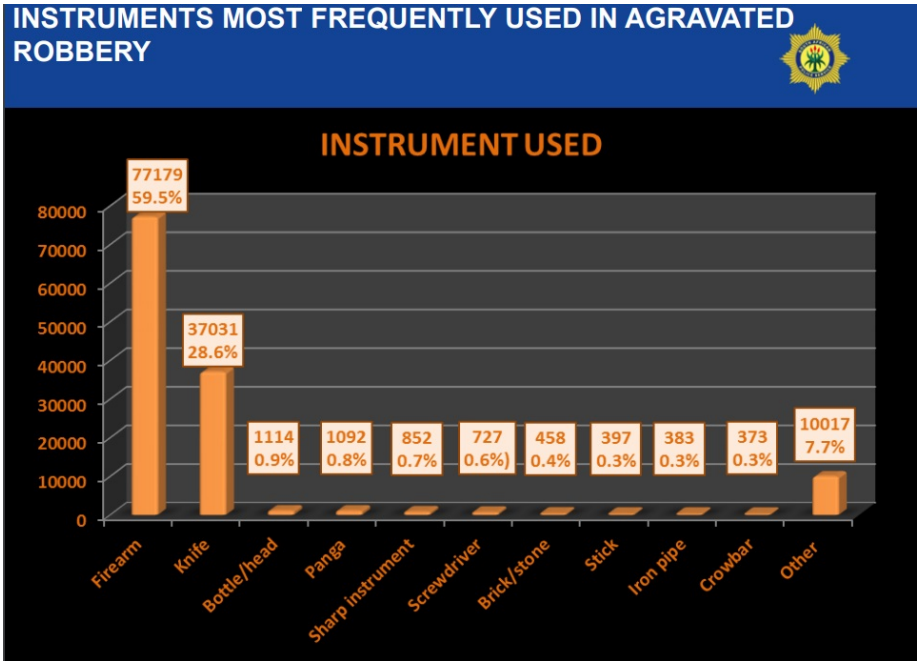


Figure 1.1: Violent crimes - statistics for weapons of choice in South Africa in 2018 [eNCA 2018]

taxi violence, cash-in-transit heists, farm attacks, and gang violence.
Southern African police commissioners found two very common firearms as the preferred weapon for crime due to their size and ability to easily conceal them [Coetzee 2004]:

1. Handgun calibres (9 x 19mm)
2. .38 special (commonly used in revolvers and some semi-automatic pistols)

Figure 1.2 shows a set of guns that were found in the possession of an unlicensed 35 year old suspect. The pistol was found in a hijacked vehicle that the suspect was driving. The two AK47s were found at the suspects home during a search conducted by the SAPS. A report compiled by Keegan [2005] looks at the nature and impact on the South African society of civilian-owned guns between 1994 and 2004. Within it is a summary of the most common types of guns used by criminals, as well as the history behind how these guns may have been circulated within society from the 1980s up to the early 2000s. A compilation of the types of guns most commonly used for criminal purposes in Southern Africa is found in Table 1.1.

Traditional security measures are still put in place to provide safety and security in vehicles on the road as well as in buildings. Burglar bars on windows, padlocked gates, and steering wheel clamps are just a few that are heavily used. However, criminals study these security systems and actively learn to breach them. For this reason, smart systems are now a growing phenomenon to further enhance safety and security, and has become a vital addition in many homes and businesses within the suburbs. Such smart security measures include closed-circuit television (CCTV), electric fencing, remote-controlled motorised gates, motion sensor beams, panic buttons, car tracker, dash cam, etc. Despite these technological enhancements in the area of safety and security, most of these systems rely on some form



Figure 1.2: Two AK47 rifles and a 9mm pistol with ammunition found through a tip-off [Writer 2016].

of human intervention to be activated. This leads to a severe delay in getting a response at the time when it is urgently needed. In the case of taking action when one or more firearms are found on dangerous persons in public areas, deep learning systems that are programmed systematically to identify the firearm and alert security personnel can be put in place to eliminate any need for human eye sighting of a firearm. These models act as the surveillance that is able to see and track the firearm within camera shot. Therefore, along with the traditional security measures already in place, the deep learning models trained for the task of firearm detection will simply automate the role of identifying a potentially dangerous situation and in turn give law enforcements a head-start in catching those in possession of firearms.

Deep learning detectors as an added measure

Deep learning detectors are a fuse of artificial intelligence and computer vision processes built on neural networks. They work by being fed raw data, and process it using a clean-up and feature extraction procedure to assist the detector to produce relevant output. In the case of firearm detection, a deep learning detector trained for this purpose is fed visual information in the form of an image upon which it performs localisation and classification to accurately pick out a firearm from the input image. For these detectors to focus solely on detecting firearms, they must undergo a training stage in which they "learn" what a firearm looks like in realistic scenarios so as to make a distinction between a firearm and non-firearm.

Properly trained detectors can be deployed in real-time and do not require any human intervention for activation. A detector trained with the right type of data with varying scenarios in which a firearm could be found would be able to accurately classify and track a firearm whether it is partially occluded or fully visible. State-of-the-art deep learning classifiers have been developed such as Faster Region-Convolutional Neural Network (Faster R-CNN) [Ren *et al.* 2015b] and Single Shot Multi-box Detector (SSD) [Liu *et al.* 2016] to accurately implement object detection in images and video feeds. Constant development is done to improve on them, and so far have shown that they are able to pick up

Pistols	**Revolvers**	**Rifles, Sub-machine Guns & Machine Pistols**	**Shotguns**
Astra	Arminius	AK-47	12 gauge Baikal
Baby Browning HP	Colt	FN FAL 7.62	12 guage Pietro Beretta
Beretta 92	Olympic 6	Galil 7.62	12 gauge Stevens
Browning HP	Rossi	Heckler & Koch G3	
Colt	Ruger	LM 4	
CZ (range)	Smith & Wesson	LM 5	
Jennings	Webley & Scott	R5	
Norinco		Skorpion	
Star		Simonov	
Taurus			
Tokarev TT-33			
Vektor (range)			
Walther (range)			
Walther P38			
Z88			

Table 1.1: Guns Most Prominent in Southern African Crime [Keegan 2005]

on the target object in a matter of seconds (as is discussed in the *Background and related research* chapter). Added to that, they have the capability of tracing the position of the target whilst not confusing it with other objects in view. Although this research is geared towards comparing the performance of detectors on images, it is possible to extend it to video feeds in which similar results can be achieved.

Deep learning detectors can be deployed to work on surveillance cameras on a 24/7 basis, therefore allowing for continuous search of firearms. Due to its computer-vision based functionality, it is unable to detect firearms that are completely hidden from sight, such as within bags or clothes. However, as soon as it is brought out into view either partially occluded or completely exposed, a deep learning detector can detect the firearm. This will significantly help to drop the crime levels simply because it is automatically and constantly on the look-out. Due to human limitations, it is not possible to always see a firearm when brought into an environment. Therefore, deep learning detectors can be used as a reinforcement to keep watch of every corner of the environment on a constant basis, and additionally offer visual details of the appearance of the firearm as well as its location.

Conclusion

Individuals in possession of firearms in areas not designated for shooting activities are indicators of danger that require action to be taken for remaining safe. The goal of this research is to compare and contrast different deep learning architectures trained for firearm

detection based on their ability to accurately detect guns in images. Firearm detectors that are able to tackle the task of identifying guns in real-time around public areas where law enforcements are unable to do so can contribute towards handling a dangerous situation.

The remaining chapters of this dissertation are organised as follows: Chapter 2 provides related research that has been previously done around firearm and object detection. Chapter 3 discusses in detail different versions of single stage object detectors with a deep-dive into their structures. Chapter 4 takes a look at the two-stage object detectors, which are the next generation of object detectors developed after the single stage models. Chapter 5 describes the methodology used to investigate the performance of deep learning models for firearm detection. Chapter 6 analyses and compares the performance of each classifier using accompanying statistics related to their accuracy and speed. Lastly, Chapter 7 closes out with a summary of the concluding points related to this research, as well as proposed future work.

Chapter 2

Background and Related Research

This chapter concentrates on the various methods used for object detection as the parent approach to firearm detection. Methods used specifically for firearm detection are also discussed.

The current chapter is organised as follows: Section 2.1 is a literature review which includes datasets that have been compiled for firearm detection, pre-processing methods, feature extraction, and classification methods. This includes reviewing of compiled datasets, pre-processing, feature extraction, and result metrics used. Section 2.2 takes a deep-dive into object detection methods including the traditional approaches, deep learning, and the hybrid approaches which all set the premise for specific object detections such as firearm detection.

2.1 Literature review

2.1.1 Generic object detection using deep learning

Over the years, deep learning techniques have developed into tools that are able to accurately identify features related to rigid objects. This feature identification allows for deep learning classifiers to specify, at varying confidence levels, what object the feature belongs to. Generic object detection can be defined as determining if an instance of an object is contained within an image, and returning its spatial location if the object is present [Liu *et al.* 2018]. Several deep learning methods for object detection have been compared and contrasted with one another. Liu *et al.* [2018] highlights that a good object detection method has high distinctiveness, robustness, and efficiency. This entails that an object can be correctly identified in an image irrespective of the number of class objects contained in one single image, or the number of variations which that object could appear as. This classification must also be done on a real-time basis with reasonable memory requirements. Faster R-CNN (a region proposal method), YOLO (You Only Look Once), and SSD (based on the regression model behind YOLO), are three classic object detection classifiers which are still being researched on for generic and specific object detection [Hechun and Xiaohong 2019].

State-of-the-art object detection frameworks have been developed with the use of regression based or region proposal methods. Regression based methods such as SSD use a single forward pass through a network to perform classification and localisation procedures for

object detection. Region proposal methods involve the selection of multiple proposed regions within an image of varying sizes in which the object may be contained. Research presented by Ren *et al.* [2015b] shows that region proposal networks like Faster R-CNN can be trained to share convolutional features which increases detection accuracy and speed of inferencing for rigid object detection.

2.1.2 Experimental evaluation on generic object detection

Zhao *et al.* [2019] evaluates different generic object detection models based on the Average Precision (AP) metric which uses a combination of precision and recall for ranked retrieval results converted into percentages [Zhang and Zhang 2009a]. Three benchmark datasets namely PASCAL VOC 2007 [Everingham *et al.* 2007], PASCAL VOC 2012 [Everingham and Winn 2011], and Microsoft COCO [Lin *et al.* 2014] have been used to train and compare the different models based on how they fair in differentiating between the generic objects.

Objects found in the PASCAL VOC datasets include different classes of animals inclusive of birds, cats, cows, dogs, horses, and sheep. The vehicle class includes boats, motor bikes, buses, cars, trains, and aeroplanes. A third class of indoor objects is made up of chairs, tables, sofas, bottles, TVs and house plants. Lastly, images of random people are present in the datasets. The COCO dataset 2017 version consists of 123 287 images. Types of recognizable objects within it can be subdivided into 9 dierent classes. These include animals, vehicles, food, furniture, electronics, outdoors, sports, dining accessories, and miscellaneous.

Comparison of performance

The results shown in Table 2.1 are based on a subset of the models discussed by Zhao *et al.* [2019]. It details outputs measured using the Mean Average Precision (mAP) which takes the average across all classes tested on. It also contains a time analysis which measures inferencing speeds and frames per second (FPS) rate. In the "Trained on" column, the values $07 + 12$ and $07 + 12 + coco$ indicate that the classifier was trained on subsets of PASCAL VOC 2007/12, or PASCAL VOC 2007/12 and COCO datasets respectively. The test set used to evaluate the different models is taken from the PASCAL VOC 2007 dataset. One conclusion drawn up based on the results on Table 2.1 states that regression based models can achieve real-time performance. However, compared with the region proposal methods such as Faster R-CNN and Region-based Fully Convolutional Network (RFCN), they consequently have a loss in accuracy [Zhao *et al.* 2019].

Model Name	Trained on	mAP (%)	Inferencing speed (sec)	Rate (FPS)
Faster R-CNN (resnet101)	$07 + 12$	83.80	2.24	0.40
Faster R-CNN (VGG-16)	$07 + 12$	73.20	0.11	9.10
YOLO	$07 + 12$	63.40	0.02	45.00
YOLOv2	$07 + 12$	78.60	0.03	40.00
RFCN (resnet101)	$07 + 12 + coco$	83.60	0.17	5.90
SSD300	$07 + 12$	74.30	0.02	46.00
SSD512	$07 + 12$	76.80	0.05	19.00

Table 2.1: Comparison of models based on mAP, inferencing speed, and FPS measures using the PASCAL VOC 2007 Test set [Zhao *et al.* 2019]

2.1.3 Object class differentiation for firearm detection use

Research that has been done centres around detecting whether or not a firearm is in view and indicating its spatial location within individual frames. This requires object classes to be generated for a deep learning system to accurately differentiate between a firearm and any other object within an image. Olmos *et al.* [2018] and Castillo *et al.* [2019] created variations of their models. One version is a binary class model represented by FIREARM and BACKGROUND classes [Olmos *et al.* 2018], while the second model version is multi-class in nature containing exactly 27 classes of varying objects other than firearms, and the 28th being the firearm class. The third model version uses 102 classes [Olmos *et al.* 2018] of objects such as aeroplanes, chair, cell phone, animals, and so on. However, out of all 3 versions, the binary class model consisting of the FIREARM and BACKGROUND classes obtained the best results. Lai and Maples [2017] used 1000 classes of objects from a database of 1.3 million images. The negative class of objects purposefully consists of objects that look similar to a gun, and can easily be mistaken for one [Kanehisa and Neto 2019]. Objects such as hair dryers and handheld power drills, as seen in Fig. 2.1, formed part of this negative class.

Traditional machine learning approaches to separate object classes from one another have also been used [Tiwari and Verma 2015]. This is done through colour-based segmentation to eliminate unrelated gun colours or objects from an image. The K-means clustering architecture separates objects based on colour into 10 clusters/classes. An approximation is done to minimize the sum of squared distance between each point and the clusters. Segmented images are formed by enhancing the colour of clusters that match with gun shades by the system while all else in contrast blanks out to create a better focus on predicted firearms.

Blob extraction is performed on segmented images from areas greater than 1000 pixels to avoid blobs from occupying small areas resulting from noise. The boundary of each blob is extracted and used to match with similar stored gun descriptors. A similarity score is calculated to measure how well each blob matches with a gun descriptor. The Nearest



(a) Power drill (b) Uzi gun

Figure 2.1: Comparison of household object to handheld gun
Sources: (a)[Mecanico] and (b)[Velocidad]

neighbour ratio algorithm calculates the similarity score to match descriptors from the blob and gun outlines as follows using sum of square difference (SSD) formula [Tiwari and Verma 2015]:

$$SSD = \sum_{i=1}^{n} (x_i - y_i)^2, \tag{2.1}$$

where x and y represent the feature vectors extracted from the two descriptors. If the resultant SSD < 0.6 then a gun match has been found within the image, otherwise the match is inconclusive.

2.1.4 Dataset formation and preprocessing

Vision-based firearm datasets for training and testing can be obtained easily from the Internet from sources such as YouTube or online databases containing firearm images and surveillance footage [Castillo *et al.* 2019].

The Internet Movie Firearm Database (IMFD) is one of the biggest databases of images containing firearms. It consists of action scenes from movies where an actor is visibly using a firearm. A total of 6585 images from the IMFD were used by Kanehisa and Neto [2019] where 90% was set apart for training, and 10% for testing. Lai and Maples [2017] only used 2735 images from the IMFD to form the firearm class, and merged it with the ImageNet dataset consisting of 1.3 million images. This merge was done with the intent of fine-tuning their database. The ImageNet dataset was also used by Olmos *et al.* [2019] and Olmos *et al.* [2018], however, Olmos *et al.* [2019] interchangeably used the COCO dataset in order to compare results obtained from the two different datasets. Approximately 3000 handgun images picked by Olmos *et al.* [2018] were added to the ImageNet dataset with the aim of reducing the number of false positives. Such guns include the revolver, automatic and semi-automatic pistols, six-gun shooters, horse pistol and derringers.

In most instances, the images forming the dataset had to go through preprocessing in order to suit the research framework, such as video frames being resized to 400×300 pixels [Tiwari and Verma 2015], on which Median-filtering was applied for de-noising. Darkening

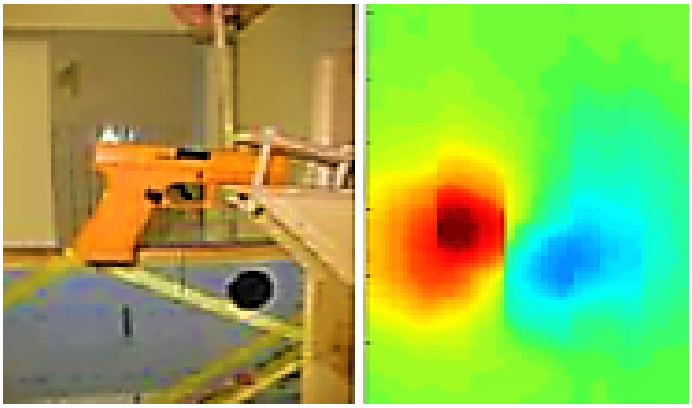


Figure 2.2: Replica-gun sample with equivalent electromagnetic image [Alqubaa and Tian 2012].

and Contrast at Learning and Test Stages (DaCoLT) is a procedure which changes the contrast of images within a certain range, done to strengthen robustness under various lighting conditions [Castillo *et al.* 2019]. After implementing DaCoLT on the images which DaCoLT, they were resized to 1920×1080 pixels. Olmos *et al.* [2018] took to resizing their images to 1000×1000 pixels to increase the frame rate therefore allowing detection to be done in near real time on video recordings, and Lai and Maples [2017] resized to 640×480 pixels to make images compatible with the TensorBox framework.

In the symmetric binocular vision approach, calculations of the disparity map were used to preselect the areas of interest in an image where actions requiring a firearm are more likely to happen [Olmos *et al.* 2019]. The electronic and computer engineering approach makes use of a system that enables Electromagnetic (EM) Field images to be constructed from measured signals [Alqubaa and Tian 2012]. Varying time and frequency features were extracted using the Fourier Transform and the Wavelet Transform to aid in implementing a robust automatic weapon detection system. Figure 2.2 displays an RGB image of a metallic gun next to its EM Field image. The EM Field image is generated by extracting only a set of features from the original image, therefore producing a reduced form of the image. A coil aligned with sensor arrays are used to pick up objects as they move across it. Two electromagnetic field signals are displayed in the RGB image in the form of coloured blobs of red and blue. The first signal is emitted by the coil from the metal detector. The second signal is generated from the picked-up features of the metallic firearm portions [Boylen 2019]. The EM field is activated when the metal comes into contact with the originally transmitted signal extracted to form a temporal EM distribution. If any metallic material is found as the object of interest moves across the sensors, the field gets distorted and the pulse response is stored into an RGB image translation along with the pulse that was sent. The RGB image in Figure 2.2 portrays the electromagnetic field emitted by the coil as a blurry blue. The retransmitted electromagnetic field from the targeted metal in response appears as a blurry red. Deeper concentrations of the hue indicate a higher density of metal.

2.1.5 Metrics used to test performance of firearm detection methods

The summary of results for Olmos *et al.* [2018], Castillo *et al.* [2019], and Kanehisa and Neto [2019] is recorded using metrics such as the number of True Positives, True Negatives, False Positives, and False Negatives. To assess the performance of the detection systems on surveillance videos, Olmos *et al.* [2018] and Castillo *et al.* [2019] used the Alarm Activation Time per Interval (AATpI) which measures time taken for the automatic detection alarm system to detect at least k successive video frames of true positives. The method used by Lai and Maples [2017] to address the problem of real-time detection is the sliding window approach of the Overfeat network, which obtained an accuracy of 89% and inferencing took 1.3 seconds.

Recording of firearm detection results can be approached using real firearms during live testing with the aim of classifying each object as either "GUN" or "NON-GUN". Alqubaa and Tian [2012] used a total of 6 different handguns made up of revolvers and automatic guns (borrowed from the police), as well as non-gun objects such as kitchen knives and scissors for live testing with the anticipation that this trained detector would be able to differentiate between what factually is a gun and what is not. The team of testers performed all tests within a controlled environment. The system is able to pick up the presence of metallic material by measuring the density and distribution of the metal in an image. This approach consists of two model versions which were compared with one another. The Electro-magnetic system combined with the Artificial Neural Network (ANN) deep learning classifier form one model which proved to have high classification rates. The second model consisted of an EM system combined with the traditional machine learning Support Vector Machine (SVM) classifier. An alert system is used to keep record of the true positives/negatives and false positives/negatives.

Dual-cameras embedded with the Block Matching (BM) and Semi-Global Block Matching (SGBM) algorithms is also used for classification [Olmos *et al.* 2019]. To evaluate the overall performance of detection, Olmos *et al.* [2019] took to using the following metric values: *accuracy* = [(TP +TN) / Total images)], *precision* which counts the true positives that were correct out of all objects that were predicted a gun, *recall* which measures the number of gun objects that were correctly picked up out of the bank of known gun objects, and the *F1 measure* which finds the average of precision and recall. This work proposed a binocular image fusion approach for reducing the number of false positives in the detection of handguns in surveillance videos by eliminating background objects.

2.2 Detection Approaches

Firearm detection is a subcategory of the broader research area of object detection which requires using computer vision techniques to recognise and localise the subject. As a result, the techniques generally used for object detection mimic that which is used for Firearm detection, making firearms the "object" to be detected. The approaches discussed below include first generation object detectors which jointly make use of machine learning algorithms and feature descriptors for detecting an object. These techniques are known as the traditional approaches. The next generation of object detectors commonly used today are the deep learning approaches which involves the use of deep networks to train a classifier for both classification and localisation. Lastly, the hybrid approach which combines more

than one method to form one hybrid architecture for detection.

2.2.1 Traditional Approaches

Traditional Machine Learning approaches tackle detection tasks by parsing relevant data into a machine learning algorithm to learn from, then making a conclusive decision based on how well the algorithm learnt from the data it analysed. It is an earlier approach to deep learning which solves computer vision problems. Examples of the firearm detection task using traditional approaches includes the Harris interest point detector and the Fast Retina Keypoint (FREAK) used to pin-point the location of a gun within an image from a dataset collected by Tiwari and Verma [2015] as an in-house gun database consisting of 65 images containing a gun. Tiwari and Verma [2015] use a combination the k-means clustering algorithm, colour segmentation, and interest point features for extracting gun-like features to detect the firearm. Interest point features from the boundary extracts are matched to stored descriptors of guns using the Sum of Square Differences (SSD) of the Nearest neighbour ratio algorithm. If the match has a similarity score greater than 50%, the alarm will be raised to signal that a gun has been detected.

Other techniques in the field include feature descriptors like the Scale-Invariant Feature Transform (SIFT) algorithm, Binary Robust Independent Elementary Features (BRIEF), and Speeded Up Robust Features (SURF) which are well-known algorithms used for object detection. The Viola Jones algorithm also falls into the category of traditional computer vision approaches. Although initially designed for face detection, its use has extended beyond face detection into the object detection space.

Viola Jones Algorithm

The Viola Jones algorithm was formulated in 2001 by Paul Viola and Michael Jones [Cen 2016]. In order for this algorithm to complete its detection process, it implements a sequence of steps, starting off with transforming the image into a summed area table which quickly calculates the sum of pixel values in a subset pixel grid [Cen 2016]. This sum reflected in Equation 2.2 is computed in both an upward direction and to the left of point (x, y) inclusive of the point using the following calculation:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{2.2}$$

The original pixel value before computing the integral image values is represented as $i(x, y)$, while $ii(x, y)$ is the complementary version on the integral image. The use of integral images allows for faster feature computation than using the original image directly. The Viola Jones algorithm extracts all possible vertical and horizontal Haar-like features from the image and stores them in an $N \times M$ matrix where $N = [(p-1) + (p-2) + ... + 1] \times [(p-2) + (p-3) + ... + 2] \times 2$ if p represents the $p \times p$ dimensions of the input image, while M is the total number of images from which features are extracted. The matrix is divided into smaller sub-matrices which are then used for training.

The next step is the use of the machine learning AdaBoost classifier made up of a weighted sum of smaller classifiers which are each a threshold for a Haar-like feature [Cen 2016]. It uses these extracted features for training the classifier. During training the smaller classifier

weights are adjusted accordingly for a given sample image when there is a misclassification until they build up into stronger classifiers that make correct predictions.

The cascade classifier is formulated by an indefinite number of stages using the strong classifiers computed by the AdaBoost algorithm. The threshold of strong classifiers at each stage is saved to formulate the cascade classifier. In stage 1, the strong classifiers are trained using the AdaBoost algorithm and used to classify samples. It is from this classification that the number of false positives (FP) and false negatives (FN) are derived. The next stage takes only the samples that were classified as positive from the previous stage for further training. The remaining samples are then used to test the trained classifier from which FP and FN are recounted. These stages are repeated in a cyclic manner until $FP, FN = 0$. The thresholds of the classifiers at this point are saved for the cascade classifier to perform object detection.

Scale-Invariant Feature Transform (SIFT)

SIFT is a traditional computer vision technique that helps to describe local features within an input image by using a 128 dimension descriptor vector. It first finds the scale-space of an input image at varying scales by using the Gaussian kernel to blur the image [Tyagi 2019]. Let $L(x, y, \sigma)$ be the scale space function for an image where x and y are points in the image and σ represents the image scale. The function used to blur each pixel of the image is as follows:

$$L(x, y, \sigma) = G(x, y, \sigma) * F(x, y) \tag{2.3}$$

Function G is the gaussian operation applied to the image represented by function F. The Difference of Gaussinas (DoG) is a set of images generated from the blurred images. It makes use of two new values for σ to derive keypoint descriptors in an image. The keypoints are a multi-dimensional set of features with an orientation, scale, and location on the blurred image which have the potential to assist in identifying an object in an image. The keypoints are then matched to keypoints in another image using the nearest neighbours algorithm to finally identify the object. The size of these keypoints consumes a lot of memory and therefore has an impact on the speed at which recognition takes place. To overcome this, dimensionality reduction is applied to reduce the keypoint descriptor size using methods such as Principal Component Analysis (PCA) or Linear Discriminant Embedding (LDE) [Bay *et al.* 2006].

Speeded Up Robust Features (SURF)

SURF detector, also known as the "Fast Hessian detector" is the computationally faster version of its predecessor SIFT due to its use of integral images (Summed area table). Its detection process is based on the Hessian matrix, and also commonly used for object detection in images [Bay *et al.* 2006]. There are various goals that had been set out for SURF to achieve, starting with the repeatability property. A number of interest points are selected at various image locations, some of which are in the same location but a different orientation therefore enhancing reliability on that image location. Another goal is the formulation of a unique descriptor. It is in the form of a feature vector containing the distribution of Haar-wavelet responses in the nearby surroundings of each interest point. They must be able to suppress detection errors and noise. These descriptors are then matched to various images in order to detect an object. To speed up the detection, 64

dimension sized integral images are used which significantly help to decrease computation time. The determinant of the Hessian matrix is used for selecting the right scale and location for the object sought after. Let the Hessian matrix be represented by $\mathcal{H}(x, \sigma)$ where $x = (x, y)$ is a point on the image I, and σ is the image scale. Using the convolutions $L_{xx}(x, \sigma)$, $L_{xy}(x, \sigma)$, and $L_{yy}(x, \sigma)$ of the second order Gaussian derivative $\frac{\partial^2}{\partial x^2} g(\sigma)$, the Hessian matrix is formulated as follows [Bay *et al.* 2006]:

$$\mathcal{H}(x, \sigma) = \begin{bmatrix} L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\ L_{xy}(x, \sigma) & L_{yy}(x, \sigma) \end{bmatrix} \tag{2.4}$$

Box filters are used to approximate the Gaussian second order derivatives on integral images for finding the appropriate scale space. In order to detect under various orientations, Haar-wavelet responses in the vertical and horizontal (x and y) direction for interest point neighbourhoods of radius size 6s, where s represents the scale at which the interest point is located [Bay *et al.* 2006].

Binary Robust Independent Elementary Features (BRIEF)

Various traditional approaches including SIFT and SURF make use of multi-dimensional vectors containing descriptors in floating point number format. As a result, a lot of memory is used up which could possibly cause processes to run slower within an application. To handle this, the floating point numbers are converted to binary strings on which Hamming distance is applied to match features in an image for finding an object. BRIEF eliminates the need to convert floating points to binary strings in order to skip steps and speed up the process even further. Instead, it finds these binary strings directly by smoothing out an input image patch and selects a set of location pairs to compare pixel intensity values. A bit vector is then created out of a small set of test responses from pairwise intensity comparisons. Equation 2.5 and 2.6 as researched by Calonder *et al.* [2010] computes these tests. If the test responses are represented by T, image patches of size $S \times S$ are represented with a P, and functions $p(x)$ and $p(y)$ are pixel intensities, then the tests can be computed as follows for each grid point (x, y) where:

$$T(P; x, y) := \begin{cases} 1, & p(x) < p(y) \\ 0, & otherwise \end{cases} \tag{2.5}$$

Taking a set of $n_d(x, y)$ location pairs, a BRIEF descriptor can then be defined in bit string format using the following equation:

$$f_{n_d}(p) := \sum_{1 \leq i \leq n_d} 2^{i-1} T(p; x_i, y_i) \tag{2.6}$$

After-which the Hamming distance is used to match the found features from the binary string BRIEF descriptors in different input images [Calonder *et al.* 2010].

2.2.2 Deep Learning

Deep learning is a branch of machine learning that consists of algorithms which have been designed based on human brain functions, specifically the neural network system [Brownlee 2020]. The concept behind deep learning is to design and train multi-layered artificial neural networks via a supervised learning approach to recognise trends in data it receives. This

training process establishes a logical relationship between the data and object class, which can be expressed as follows: given vector $X = (x_1, x_2, ..., x_n)$ where X consists of n sample inputs each associated with a label $Y = (y_1, y_2, ..., y_n)$, the algorithm must find a function that relates each sample input to a known label in the form $f : X \rightarrow Y$ [Egiazarov *et al.* 2019]. The performance of deep learning approaches based on scalability and accuracy in tasks like semantic segmentation, object detection and simultaneous localization and mapping has surpassed its traditional counterpart [O'Mahony *et al.* 2019], therefore making it a common choice for object detection tasks in the computer vision space. Convolutional neural networks (CNNs) are a well-known example of techniques that are commonly used for the general object detection task, and are a perfect fit for efficiently developing a firearm detection model that is required to undergo a training process using input images containing one or more firearms. Research has shown that the firearm detection methods incorporated today mostly make use of deep learning approaches.

The CNN is a type of neural network in deep learning that has been designed to handle tasks such as segmentation, human pose estimation and video classification. Another such task is the object recognition task of identifying objects based on their spatial location in an image and object class. In order to learn how to identify thousands of objects from millions of images, a deep learning model is needed with a huge and complex learning capacity which CNNs posses due to the ability to control their layer depth depending on what is required to complete the task [Krizhevsky *et al.* 2017]. Furthermore, within its deep layers prominent features such as consistent object patterns of continuous distribution, as well as object texture are learnt by the CNN [Choi *et al.* 2016]. Research has shown that the CNN is now becoming one of the most commonly used methods for analysing graphical data. Variations of Convolutional Neural Networks have been used by Olmos *et al.* [2018], Castillo *et al.* [2019], and Kanehisa and Neto [2019] for classification and detection. The deep CNN used by Olmos *et al.* [2018] is the VGG-16 network pre-trained on the ImageNet dataset. This network was fine-tuned using the dataset gathered by Olmos *et al.* [2018] consisting of a total of 3000 images of automatic/semi-automatic pistols, horse pistols, revolvers, six-gun shooters, and derringers. Both the sliding window approach and the region proposal approach were compared and contrasted on the VGG-16 network based on their speed of inferencing and accuracy in detecting a firearm. The sliding window approach had more classes to differentiate between than the region proposal method. The classifier was applied in steps of 60×60 pixels to windows of 160×120 pixels of each input image [Olmos *et al.* 2018]. The region proposal approach contained a wider variety of data in the training set due to the nature of each approach. Faster-RCNN is the classifier choice combining the selective search approach with VGG-16 based classifier. The best performing approach used by Olmos *et al.* [2018] is the region proposal method that makes use of the Faster-RCNN classifier, and as a result has been used in this research as one of the classifiers to be compared and contrasted. The approach used by Castillo *et al.* [2019] makes use of a combination of region selection methods to generate proposed regions from the input image, and CNNs to analyze and classify each proposed region for cold steel weapon detection in surveillance videos. The three CNN-based classification models used are ResNet, Inception, and Inception-ResNet-V2.

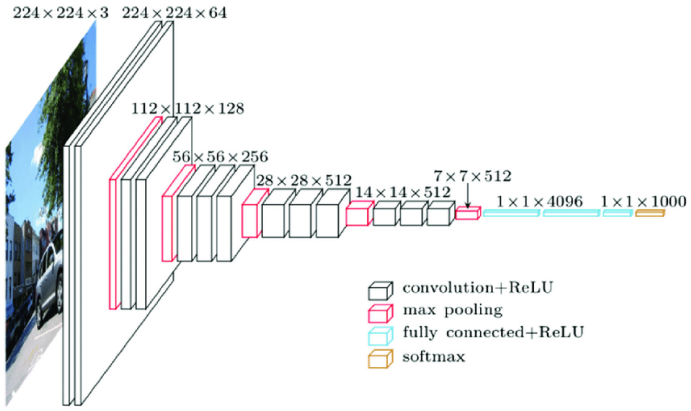


Figure 2.3: VGG16 architectural design [Nash *et al.* 2018].

VGG-16

VGG16 is one of many convolutional neural networks that have been formulated for computer vision tasks. It became well-known at the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC), an event that draws developers worldwide to compete in a visual recognition competition. It is known to host benchmark classifiers in the object detection and computer vision fields. The aim is to produce and train models using the provided dataset at the event over a period of four months. It is from this event in 2014 when the Visual Geometry Group from Oxford University placed 2nd place after presenting the VGG16 network [Nash *et al.* 2018].

The network is made up of 16 layers each with their own weight parameters to apply on fed-in input data. It works by transforming the input image into a multi-class vector depending on the number of unique objects being detected. In the case of Figure 2.3, the dataset provided by ILSVRC contained 1000 classes, therefore having VGG16 produce a $1 \times 1 \times 1000$ vector. Rather than having multiple number of hyper-parameters, the network consists of 13 convolutional layers made up of 3×3 filters with a convolutional stride of 1 pixel, 5 max pooling layers made up of 2×2 filters and a stride of 2 pixels [Zhang *et al.* 2015]. The filters mentioned assist in detecting the spatial pattern found within each input image by keeping track of changes in intensity as it slides across the image. At each stage of the network, the filters applied help to reduce the number of parameters carried forward therefore reducing computation [Thakur 2019]. The movement of these filters is determined by the stride parameter, which regulates by how many pixels a filter moves across the image at a time. The filters perform max pooling in specific layers, which is the process of extracting the maximum pixel value in areas of the feature map where convolution is applied, which is then passed forward. This structure of having convolutional and max pooling layers following after one another extends throughout the network. It ends with three fully connected layers known as the "dense layers", followed by an output in the form of softmax, a process which assigns decimal probabilities to each class, and sum up to 1.0 when added together. VGG-16 make use of gradient descent with backpropagation for training. Similar structures making use of convolutional and max pooling layers can be found in the Fast/Faster R-CNN and YOLO models.

One of the challenges faced by CNNs is the vanishing gradient problem. The more lay-

ers added to the network, the more the gradient of the loss function tends to zero. The gradient is back-propagated causing repeated multiplications which makes the gradient to become extremely small. This in turn decreases successive updates to the weights in the network therefore making the model almost untrainable. This results in an overall performance saturation that occurs due to lack of training. The hidden layers make use of the Rectified Liner Unit (ReLu) activation function to overcome the vanishing gradient problem, where each node implements the function to transform the summed weighted input to an output. The function is defined by $f(x) = max(0, x)$ [Albani *et al.* 2017]. It is formulated to equal zero for all negative valued inputs, and form an identity function for all positive valued inputs, therefore producing significantly large derivatives and allowing for faster convergence. Due to the enormousness of the network, there are approximately 138 million parameters contained in it, therefore making this network a heavyweight and slow compared to its counterparts. It is used as a base network by architectures such as Faster R-CNN for object detection tasks.

ResNet

The Residual Network architecture, known as ResNet, is a deep residual learning CNN framework consisting of stacked residual blocks used for training hundreds of layers in a convolutional network [Le Huy Hien and V.H. 2020]. Each block consists of "residual" or "shortcut" connections that skip layers in between to speed up the training process whilst retaining accuracy. The formulation of each block as portrayed by Le Huy Hien and V.H. [2020] is as follows:

$$y_i = h(x_i) + A(x_i,\ W_i),\ x_{i+1} = f(y_i) \tag{2.7}$$

where variables x_i and x_{i+1} represent the input and output respectively of the i^{th} residual block, $h(x) = x_i$ represent the identity mappings, A is the residual function applied upon the product of inputs x_i and weight W, and function f represents the ReLu function applied to output y. In order to counter the vanishing gradient problem caused from addding layers to create a "deeper" network, ResNet has been developed to allow stacked identity layers fit a residual mapping [He *et al.* 2016]. This can be formulated with a function where $H(x)$ represents the underlying mapping that is required, the original mapping is reshaped into $F(x) + x$ through the residual connections that skips at least one identity layer (see Fig. 2.4), and the stacked non-linear layers are represented by $F(x) := H(x) - x$ where x denotes the inputs for the first layer of those stacked to fit the underlying mapping $H(x)$. The stacked layer outputs are derived from the shortcut connection identity mappings. The advantage of these identity mappings is that they add no form of computational complexity to the network, which in turn optimizes the already deep network.

Variants of ResNet have been developed since its inception, such as ResNet version 2 (ResNetV2), where the main difference in Version 2 is seen in the layer arrangement within each residual block. Just before each weighted layer, Batch Normalization (BN) is implemented, followed by the ReLU activation then ending with the 2D convolution (Conv2D). This forms a Residual block of BN-ReLU-Conv2D layers of 1×1, 3×3, and 1×1 following one after another.

ResNeXt consists of a more complicated structure than ResNetV1 and ResNetV2, in the sense that layers of ResNeXt are stacked in parallel format and concatenated at the end as

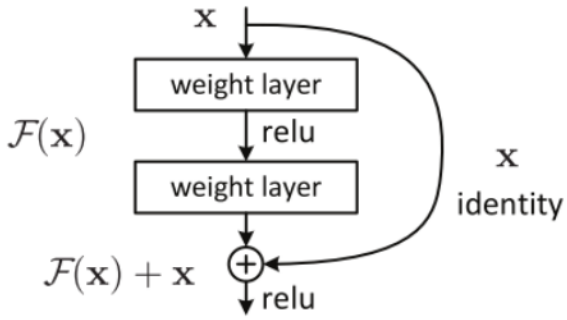


Figure 2.4: Residual Learning - shortcut connection [He *et al.* 2016].

opposed to the serial sequential structure of the earlier ResNet versions. It contains multiple paths per residual block of stacked identity mappings. The number of independent paths contained per residual block is represented by the newly introduced hyperparameter called the "cardinality" C. Each path retains the same topology but produces its own output. The convolution outputs are then merged together by addition to formulate a resultant output. Even though the complexity of ResNeXt is significantly higher, the training error compared to the older ResNet versions was lower, therefore resulting in better performance. ResNeXt has been incorporated into the deep learning model Mask R-CNN as its base network to carry out instance segmentation and object detection [Xie *et al.* 2017]).

The bipolar morphological ResNet (BM-ResNet) model, more complex than its predecessors, is formed using a bipolar morphological neural network. The BM-ResNet model has been tested in various computer vision scenarios including text field recognition, and can be extended to object based recognition due to its design of reading visuals as an input to produce relevant outputs. It is designed to separate positive and negative data by placing them in their own processing paths in the network [Limonova *et al.* 2020a]. To increase computational efficiency, each convolutional operation of multiplication has been replaced with an addition and maximum/minimum operation [Limonova *et al.* 2020b]. The bipolar nature of the network stems from the two computational paths that represent the excitation and inhibition processes of a neuron. To fine-tune the network during training, the training process involves modifying convolutional fully connected layers from the beginning to the end of the network in a sequential manner. After each completed modification, the modified layer along with its weights are frozen while other layers undergo training in order to bypass possible issues that could arise when directly training bipolar morphological layers or neurons. Equations 2.8 up to 2.13 as documented by Limonova *et al.* [2020a] relate to various portions of the formulation of neurons. The operations performed by a classic (non-bipolar morphological) neuron can be expressed as follows:

$$y(x, w) = \sigma\left(\sum_{i=1}^{L} w_i x_i + w_{L+1}\right) \tag{2.8}$$

In this classical neuron function y, the variable x represents the input vector with length L, w represents the weight vector with a length of $L + 1$, and the activation function is represented by σ. This differs in complexity from the bipolar morphological neuron

operation which is expressed as follows:

$$y_{BM}(x,V,v) = \sigma\Bigg(\exp \max_{j=1}^{L}(\ln x_j^+ + V_j^+) - \\ - \exp \max_{j=1}^{L}(\ln x_j^+ + V_j^-) - \exp \max_{j=1}^{L}(\ln x_j^- + V_j^+) + \\ + \exp \max_{j=1}^{L}(\ln x_j^- + V_j^-) + v \Bigg) \tag{2.9}$$

The equation above is a non-linear activation function of the bias value v, and input vector x and weight vector V both of size L, where:

$$x_j^+ = \begin{cases} x_j, & x_j \geq 0 \\ 0, & x_j < 0 \end{cases} \tag{2.10}$$

$$x_j^- = \begin{cases} -x_j, & x_j < 0 \\ 0, & x_j \geq 0 \end{cases} \tag{2.11}$$

The positive and negative parts of input vector x are treated as two different computational paths representing the excitation and inhibition processes consecutively in the network. The weight vectors V_j^+ and V_j^- are treated and trained separately as independent weights, and are represented as follows:

$$V_j^+ = \begin{cases} \ln w_j, & \text{if } w_j > 0 \\ -\infty, & \text{otherwise} \end{cases} \tag{2.12}$$

$$V_j^- = \begin{cases} \ln|w_j|, & \text{if } w_j < 0 \\ -\infty, & \text{otherwise} \end{cases} \tag{2.13}$$

During training of BM-ResNet, each convolutional and fully connected layer has its weights w modified by being replaced with the bipolar morphological weights as seen in Equations (2.7) and (2.8).

Inception

The Inception model, as described by Lin *et al.* [2013] and Shaikh [2018], has been built to tackle the object detection task in an efficient manner by increasing the width and depth of the search while at the same time preserving computational complexity. The name "Inception" is derived from the article written by Lin *et al.* [2013] in which the deep network structure "Network In Network" (NIN) made up of micro neural networks using linear filters to scan input images is discussed. This article paired with the movie *Inception* (of 2010) in which the phrase "we need to go deeper" formed the basis behind the Inception neural network architecture. Going deeper in the case of this model is referring to the increased number of levels in the network as well as an increase in the number of filters operating simultaneously per level to produce a concatenated output [Szegedy *et al.* 2015]. The model has been designed to have convolution filters of size 1×1 for dimension reduction, and more expensive filters of size 3×3 and 5×5 existing in each layer. The outputs from these convolutions are used to form one single output vector as an input to the next

layer. Inserted within the network at occasional stages are max-pooling layers to reduce the grid size. The first version of the Inception model known as "GoogLeNet" enhances the ability to increase the width at each stage of the network without causing huge additional computational complexity. This is due to the dimension reduction performed which covers up the number of filters applied per layer. Secondly, concatenating the different filter outputs into one as the next layers input allows for the network to learn visual object features at different scales all at once. In the ILSVRC 2014 object detection challenge, each model underwent training to detect 200 different object classes. Recognition consisted of labelling the objects correctly based on the ground truth class, and indicating its location on the image using a bounding box which must have a minimum 50% overlap with the ground truth box. The results obtained by each model is based on the mean average precision value (mAP). The selective search approach was used in conjunction with the multi-box predictions in order to improve the region proposals and obtain higher object bounding box recall. As a result, GoogLeNet tied in first place with UvA-Euvision with an mAP of 43.9%.

The second version of Inception (Inception v2) is a 42-layer network designed to counter the challenges found in the first version such as the "representational bottleneck", also commonly seen in other deep learning networks. This occurs when dimensional alterations are applied to input images in the feed forwarding process. Various bits of information like correlation structure are discarded from the input as it passes through the network, therefore causing the output to be a rough estimate based on the information it has received [Szegedy *et al.* 2016]. To avoid the representational bottleneck effect, both the convolution and pooling layer box are set at stride 2, and the grid-size is reduced simultaneously while the filter banks expand via concatenation, resulting in cheaper computation and loss of less feature information. The Inception v2 also aims at speeding up the training time experienced by the first model by increasing activations per tile within the network. One of the most obvious changes that has been implemented is factorizing convolutions at a suitable level with large filter sizes to reduce computational cost. Szegedy *et al.* [2016] argues that large filters such as the 7×7 and 5×5 are unnecessarily expensive computationally. The convolution of size 5×5 is $25/9 = 2.78$ times more expensive than the 3×3 convolution applied across the same feature map grid. To tackle this, the 5×5 convolution has been replaced by a two-layer convolution design of 3×3 convolutions. Figure 2.5 visualizes the changes made from the first inception model to the the second version based on the convolution factorizations made, in which the 5×5 convolutions have been replaced with two 3×3 convolutions. To further cut on computational costs, factorization is explored even further by breaking down $n \times n$ convolutions into an equivalent made up of two layered convolutions in the form of $n \times 1$ and $1 \times n$. The computational savings gained increases as n attains a higher value. This method of factorization is best applied to medium or big sized $m \times m$ feature maps where $m \geq 12$.

The third version of this model, known as Inception v3, incorporates the use of auxiliary classifiers which assist with convergence of deep networks. They are attached to layers before the end of each network to improve convergence by pushing gradients to lower layers. This helps to prevent the vanishing gradient problem experienced in deep networks. Batch normalization is then applied to the fully connected layers of the auxiliary classifiers. Factorization of convolutions is further applied to the 7×7 kernels into multiples of smaller 3×3 kernels to further decrease computational expense. The concept of label-smoothing regularization (LSR) is also used, in which a ground-truth label distribution is applied to

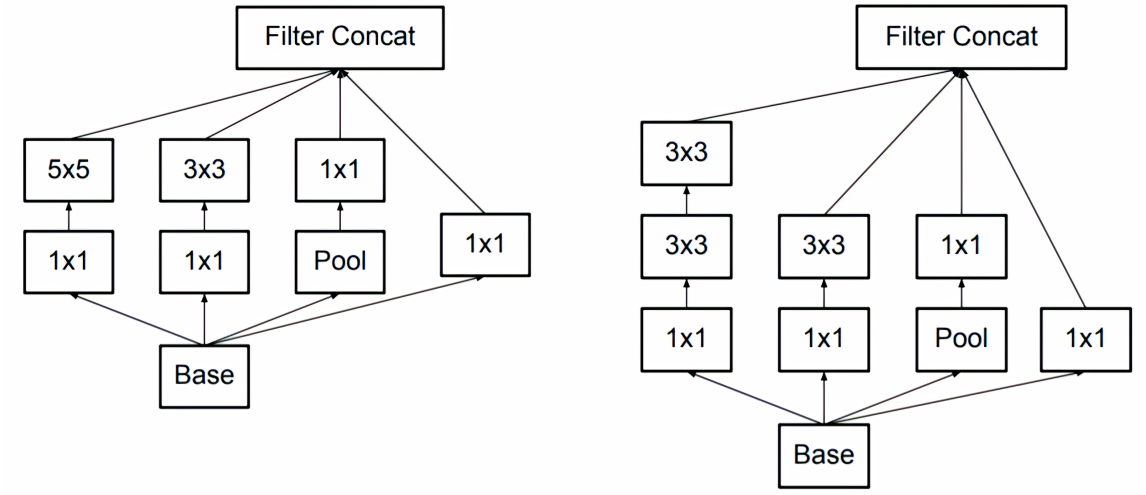


Figure 2.5: Inception model version 1 (left); Inception model version 2 (right) [Szegedy *et al.* 2016].

prevent overfitting. This means that the confidence level is reduced when selecting a class label in order to regularize the model. This is formulated in equations 2.14 to 2.15 [Szegedy *et al.* 2016]. Let $u(k)$ represent a distribution of predicted labels where $k \in \{1...K\}$, the smoothing parameter is represented by ϵ, and the ground-truth label distribution is $q(k)$. Furthermore, let x be a training example, y is a ground-truth label, and $\delta_{k,y}$ be the Dirac delta where $\delta_{k,y} = 1$ for k = y otherwise 0. The label distribution where $q(k|x) = \delta_{k,y}$ is replaced with the following:

$$q'(x|k) = (1 - \epsilon)\delta_{k,y} + \epsilon u(k) \tag{2.14}$$

This equation is further formulated as a distribution of label k where $k = y$, and the uniform distribution $u(k) = 1/K$ to obtain the LSR equation as follows:

$$q'(k) = (1 - \epsilon)\delta_{k,y} + \frac{\epsilon}{K} \tag{2.15}$$

The advantage of the Inception model is its highly adjustable nature, implying that it is possible to change or increase the number of filters within each layer without negatively impacting the quality of network performance [Szegedy *et al.* 2017]. For this reason, multiple version of the Inception model have been developed. Inception version 4 introduces a modified structure starting out with the "stem": the initial layer that performs a set of operations in the network. This is followed by three inception blocks spread out in the network labelled as inception module A with a 35×35 grid, module B with a 17×17 grid, and module C with an 8×8 grid. Each have their own unique networking structure that aims to use multiple filter sizes simultaneously in obtaining a concatenated output for the next layer. Also introduced in Inceptionv4 is the Residual modules A and B which reduce the width and height grid size from 35×35 to 17×17 and from 17×17 to 8×8 consecutively. Inception modules A and B are followed by Reduction modules A and B for dimension reduction of the concatenated outputs. Inception module C is followed by the average pooling layer and softmax from which a final output is produced.

2.2.3 Hybrid Approach

Researchers such as Szegedy *et al.* [2017] have ventured into mixing different neural networks together to produce a hybrid network with the idea that the strengths of one network will make up for the weaknesses of the other, therefore creating a fortified hybrid network. Examples of this include the hybrid inception models known as Inception-ResNet v1 and v2 which contain residual connections that add convolution outputs in the inception modules to the inputs. This helped to increase the training speed of the overall architecture for object recognition. Both versions share very similar structures by containing inception modules A, B, and C as well as reduction modules A and B. However, the structure of their stems differ in complexity and order of operations. The Inception-ResNet v1 stem structure is more uniform in nature and less complex as it consists of 3×3 and 1×1 convolutions with a max pooling layer in between the convolutions, whereas the Inception-ResNet v2 stem follows a similar structure to that which pure Inception v4 has [Szegedy *et al.* 2017], splitting into two separate paths for max pooling and convolution operations before implementing filter concatenation. As a result, the computational cost of Inception v4 and Inception-ResNet v2 are similar and higher than that of their counterparts Inception v3 and Inception-ResNet v1 which also share similar computational cost. One of the main differences between the Inception-ResNet models and the pure Inception models is the point at which batch normalization takes place. Batch normalization is a transformation process in which sets of inputs provided into the network undergo standardization in order to speed up the training process for fast approach to convergence [Saxena 2021]. In the ResNet variants, batch-normalization is performed on the traditional layers instead of on top of the summations as with the pure Inception models in order to increase the number of inception modules overall in the network. .

Chapter 3

Single Stage Object Detectors

In this chapter, different versions of single stage object detectors are discussed in detail. This includes an overview of their structure, and the procedures followed to carry training process of the classifier when fed input images.

This chapter is outlined as follows: Section 3.1 provides a summary of the general processes used by single stage object detectors. Section 3.2 details the SSD classifier by discussing its structure and internal processes for performing detection. Section 3.3 presents the YOLO classifier by discussing its network structure and each step performed for object detection. Section 3.4 provides details of YOLOv2 by looking at improvements that were made to the previous version. Section 3.5 contains the changes that were made to generate YOLOv3.

3.1 Overview of single stage object detectors

Since the inception of object detection, various computer vision techniques and deep learning models have been developed with the aim of accurately locating and classifying rigid objects based on graphical information. Single stage object detectors are one of two types of object detection architectures which solves the detection task as a regression problem using a single neural network [Soviany and Ionescu 2019]. These type of detectors are highly focused on inferencing speed by going through a single forward pass through neural network [Hollemans 2018] and running detections over multiple samplings of possible object locations within a single image at one go [Weng 2018]. Both multi-stage and single stage object detectors make use of Mean Average Precision (mAP or Average Precision) to measure the performance of the classifiers. It is a function of precision and recall for ranked retrieval results that calculates the mean of the precision scores after each relevant piece of information is retrieved [Zhang and Zhang 2009b], in this case from an image. Let N refer to the total number of input images, and AP_i be the average precision as a product of precision and recall at input i. Then the formula for calculating the mAP is as follows:

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{3.1}$$

For a prediction to be made, single stage object detectors predict the Intersection over Union (IoU) between the ground truth box (created from annotating images from the training set) and the anchor box (a spatial location estimated output made by the trained classifier). This is measured as a score between 0 and 1, where 1 represents 100% anchor

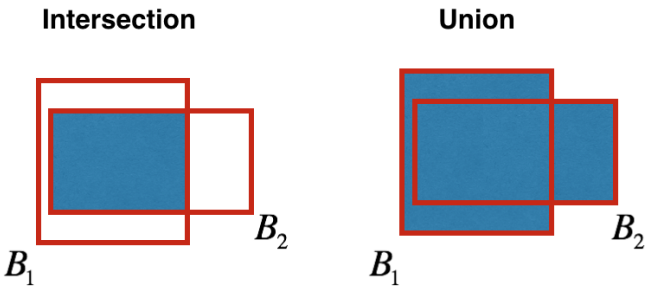


Figure 3.1: Example of $B_1 \cap B_2$ (left) vs. $B_1 \cup B_2$ (right)

match between the regressed box and the ground truth box. Figure 3.1 shows an example of the overlap between bounding boxes B_1 and B_2 forming an intersection and a union in the blue shaded portions. The IoU can be calculated using the following formula:

$$IoU = \frac{B_1 \cap B_2}{B_1 \cup B_2} \quad ; \text{where } 0 \leq IoU \leq 1 \tag{3.2}$$

This method works efficiently when one sample of the object is contained in the image as in Figure 3.2 where the ground truth box is in green and the prediction is a red bounding box with a confidence score of 0.8 attached to it. Only one firearm is present in the image, and as a result the classifier is able to pick up the relative position of the object, resulting in an overlap of both the ground truth and predictor box. The IoU has been designed to predict the location of all detected objects using one bounding box. As a result, when there are multiple appearances of the object in a single image, the bounding box is placed approximately in the middle of all the located objects. If N represents the number of bounding boxes produced at the prediction stage, then having $N > 1$ bounding boxes will still result in all predicted bounding boxes placed in the middle. Single-stage object detectors such as SSD, YOLO, and RetinaNet have been designed to overcome this challenge by splitting the image into a regular grid of locations in which each detector is assigned to estimate the object location, class of object, and confidence score [Poirson *et al.* 2016]. A convolutional predictor is used to generate the final output in the form of a tensor consisting of the bounding box coordinates and the class probabilities in the form of a confidence score value as seen in Figure 3.3 below [Noh *et al.* 2018].

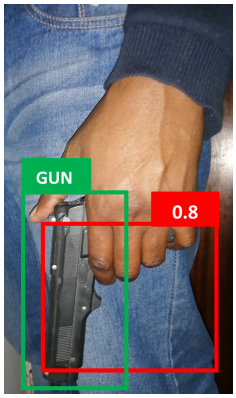


Figure 3.2: Firearm detection output predicted with red bounding box, and green ground truth box containing a firearm as the target object.

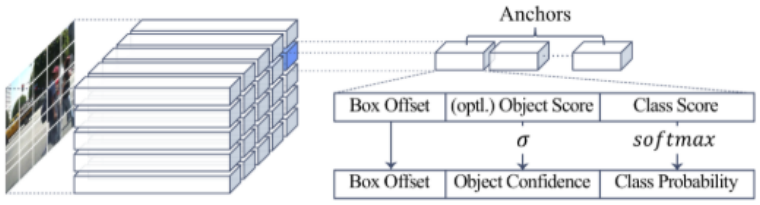


Figure 3.3: Output tensor structure [Noh *et al.* 2018]

The image is split into a spatial grid where each block is made up of anchors. Within each anchor is a predicted box offset value that represents the probability of having overlapped the object, an object score which is converted into a confidence value representing its level of confidence for containing an object, and the class score which softmax is applied on to generate the class probability. In the case of a multi-class model, the object class with the highest probability would be assigned to that anchor box.

3.2 Single Shot Multibox Detector (SSD)

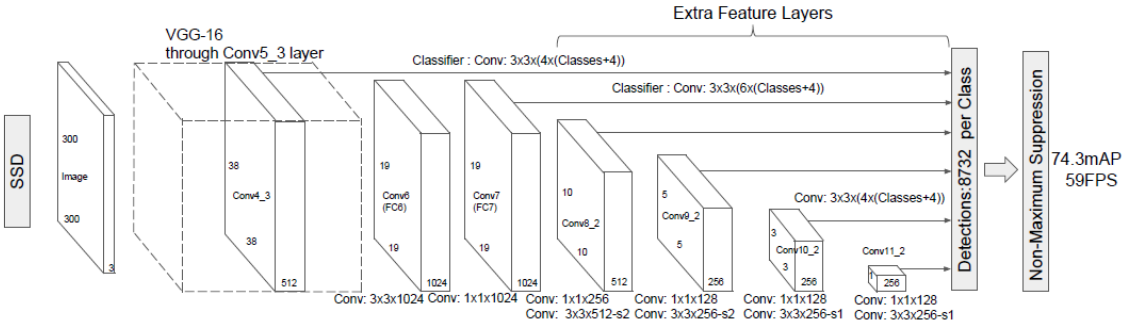


Figure 3.4: SSD structure with an input image of size $300 \times 300 \times 3$ [Liu *et al.* 2016]

SSD is an object detection approach that was released in the year 2016 as a one stage object detector. It makes use of images annotated with ground truth bounding boxes analysed for training. It is structured to perform the tasks of localization and classification of an object in a single forward pass (hence the name, single shot) through a CNN. This network produces a series of default bounding boxes, also known as anchor boxes, of fixed sizes (hence the name, multibox). Each box is scored based on whether the target object to be detected is found within it. This score is translated into a probability value which describes the level of confidence that the detector has found the object in a specific location in the image.

The improvements that were proposed at the inception of SSD was to use convolutional filters in order to predict the object classes and offsets. This is done using different kernels for various locations in the image. These kernels are applied to multiple feature maps in order to perform the detection task [Liu *et al.* 2015]. Several feature maps which span the input image are set at different scales.

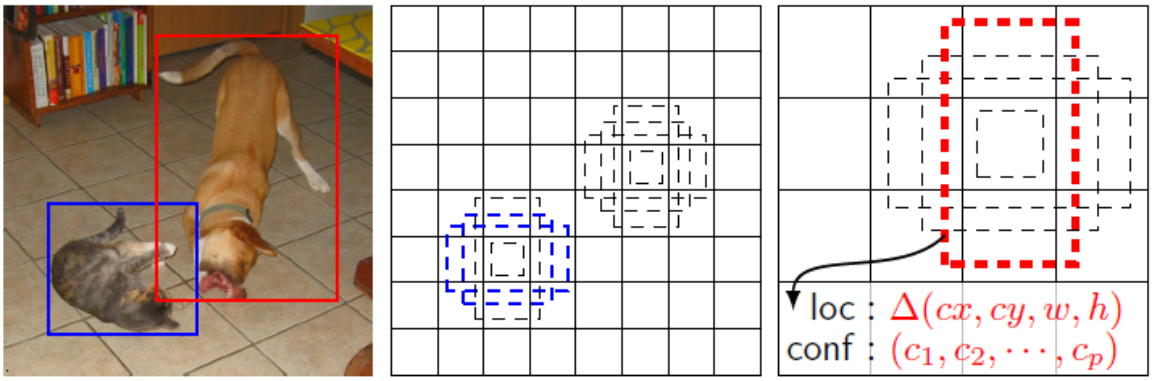


Figure 3.5: Input image of a cat and dog within their own ground truth box, alongside an 8 x 8 and a 4 x 4 feature map with default boxes predicting the localization and classification output. The blue boxes correspond to the cat, while the red corresponds to the dog [Liu *et al.* 2016]

3.2.1 SSD Structure

As at the release of SSD, the early network layers called the base network, were truncated layers built from the VGG-16 architecture designed for high quality image classification. Convolutional feature layers are added at the end of the base layer which progressively decreases in size as seen in Figure 3.4 where the input image starts of at size 300×300 and undergoes dimensionality reduction after each convolution. Connected to the base layer is a series of feature layers which progressively perform dimensionality reduction upon feature maps. As a result, the size of the feature maps decreases as they are passed from one layer to the next to allow for predicted detections at multiple scales. A set of convolutional filters of size $3 \times 3 \times c$ (where c = channels) for each layer is applied upon the various locations of the feature maps to predict parameters for a potential detection. A total of $(p+4)k$ filters are applied at the various locations on the feature map of size $u \times v$, where p = confidence score of object class presence, and k = total number of default boxes per location. This results in $(p+4)kuv$ outputs for each feature map.

3.2.2 Default Boxes

A small set of overlapping default bounding boxes of different dimensions are produced when the IoU with the ground truth boxes is greater than 0.5. They are placed in various locations of the feature maps and used to predict the presence of the target object within it as well as the shape offset. This multibox principle of default boxes is designed to increase the chances of locating and detecting objects of various sizes and orientations. Figure 3.5 describes how the feature maps are applied to the image to perform the localization and classification task.

The assigned default boxes are scaled accordingly depending on the size of the feature map they reside in. The allocated scales range between 0.2 and 0.9 with multiple aspect ratios per scale. Assume that for a prediction to be made, a total of n feature maps are needed. The scale (α) for the i^{th} feature map in the network is then calculated using minimum and

maximum scale values α_{min} and α_{max} consecutively in the formula below [Liu *et al.* 2015]:

$$\alpha_i = \alpha_{min} + \frac{\alpha_{max} - \alpha_{min}}{n-1}(i-1), \text{ where } i \in [1, n] \tag{3.3}$$

The different classes of objects that the detector has been trained to locate and classify is represented in a Confidence vector (C) produced by the default boxes, and the coordinates of the bounding box containing the object is represented in a Localisation vector (L). These vectors can be represent as follows:

$$C = \begin{bmatrix} c_1 & c_2 & . & . & . & c_p \end{bmatrix}^T \quad ; \quad L = \begin{bmatrix} c_x & c_y & w & h \end{bmatrix}^T$$

In the confidence vector, variables c_1 to c_p represent the different classes of objects which the classifier has been trained to identify. Each variable takes the value 1 if the object is present or 0 otherwise. In the localisation vector, c_x and c_y are the centre point coordinates of the bounding box containing the object. This centre point is measured with respect to the centre of the feature map cell it is contained in, while variables w and h represent the width and height of the box. The network will produce a variable p which is an estimated probability value that represents the confidence of having found the target object within the bounding box.

3.2.3 Loss Function

The loss function L_f applied for SSD is the weighted sum of the confidence loss (L_{conf}) and localisation loss (L_{loc}), where N refers to the number of matched default boxes to ground truth boxes [Liu *et al.* 2015]. The general formula is as follows:

$$L_f(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g)) \tag{3.4}$$

Localisation Loss

This is the smooth $L1$ loss between the predicted bounding box (l), and the ground truth box (g). It focuses on penalizing mismatches from the positive match set in order to find matches that are as close as possible to the ground truth box location and dimensions. If $x^k_{i,j}$ refers to either a positive (1) or negative (0) match between the i^{th} default box (d) and j^{th} ground truth box in the k^{th} object class, then the formula for L_{loc} can be written as follows:

$$L_{loc}(x, l, g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx, cy, w, h\}} x^k_{i,j} smooth_{L1}(l^m_i - \hat{g}^m_j) \tag{3.5}$$

$$\hat{g}^{cx}_j = (g^{cx}_j - d^{cx}_i)/d^w_i \qquad \hat{g}^{cy}_j = (g^{cy}_j - d^{cy}_i)/d^h_i$$

$$\hat{g}^w_j = \log\left(\frac{g^w_j}{d^w_i}\right) \qquad \hat{g}^h_j = \log\left(\frac{g^h_j}{d^h_i}\right)$$

Confidence Loss

This measures the softmax loss over multiple class confidence values c [Liu *et al.* 2015]. Default boxes without a match to a ground truth box are negative and therefore contribute

towards the confidence loss. The L_{conf} formula can be written as:

$$L_{conf}(x,c) = -\sum_{i \in Pos}^{N} x_{i,j}^{p} \log(\hat{c}_i^p) - \sum_{i \in Neg} \log(\hat{c}_i^0) \text{ where } \hat{c}_i^p = \frac{exp(c_i^p)}{\sum_p exp(c_i^p)} \tag{3.6}$$

3.2.4 Non-Maximum Suppression



Figure 3.6: Example of NMS applied to an image with multiple default bounding boxes [Hosang *et al.* 2017]

During a forward pass through the SSD network, a large number of bounding boxes are generated which point at the same object in an image. In order to obtain the final detection results, a process referred to as Non-Maximum Suppression (NMS) is applied to remove the excess bounding boxes. This is done using the estimated confidence score values. Boxes which have a confidence loss value less than 0.01 and an IoU less than 0.45 are eliminated, leaving the default boxes which are more closely matched to the ground truth boxes [Hosang *et al.* 2017]. This will allow for a final bounding box detection. Figure 3.6 displays what the image looks like with default boxes on it when detecting people before NMS is applied, and the result of applying NMS upon it. The process can be explained using Pseudo code from Bodla *et al.* [2017], where a list of N initial detection boxes and corresponding scores per object are represented using set $B = b_1, ..., b_N$ and $S = s_1, ..., s_N$ consecutively. Let N_t be the NMS threshold, and D be the set of final detections containing maximum score M. Then the pseudocode to perform NMS upon an image can be written as follows:

Algorithm 1: Non-Maximum Suppression [Bodla *et al.* 2017]

$D \leftarrow \{\}$
while $B \neq empty$ **do**
    $m \leftarrow argmax\ S$
    $M \leftarrow b_m$
    $D \leftarrow D \cup M; B \leftarrow B - M$
    for b_i *in* B **do**

7: **if** $iou(M, b_i) \geq N_t$ **then**
8: $B \leftarrow B - b_i; S \leftarrow s_i$
9: **return** D, S

Lines 7 and 8 are where NMS is performed to reduce the number of bounding boxes around an object. The detection is selected by choosing M from set S and placing it in the initially empty set D. Also removed from set S are all boxes with a value greater than N_t with M in set B. This is repeated in a cyclic manner until nothing more remains in set B. At this point, set D now contains all required bounding boxes in which the object has been detected, and the unnecessary repeats of boxes have been discarded.

3.2.5 Deconvolutional Single Shot Detector (DSSD)

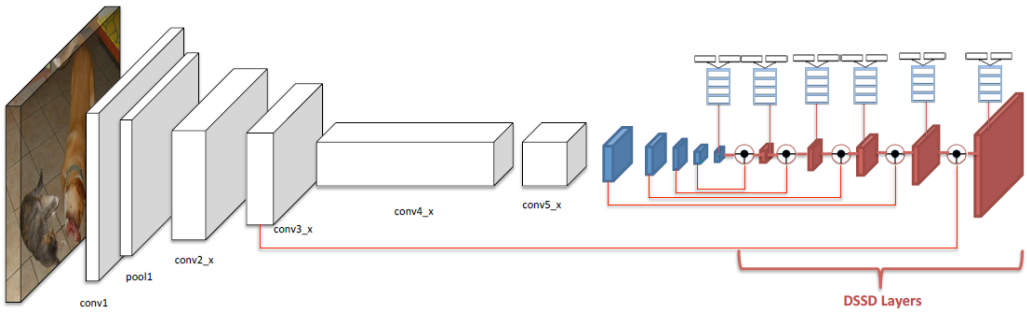


Figure 3.7: Structure of DSSD with additional deconvolutional layers [Fu *et al.* 2017]

Later versions of the SSD structure replaced the base network built on VGG with the Residual Network (ResNet) 101 network [Fu *et al.* 2017] to improve on the accuracy of the classifier. It is able to go 101 layers deep where VGG allows for 19 layers. As seen in the original SSD framework, the feature maps are reduced in size as they are passed from on feature layer to the next. This has a negative impact on detecting small objects in an image. Therefore, deconvolutional layers are added at the end of the network as seen in Figure 3.7 to enhance the resolution of the feature maps in order to scale-up the size of smaller objects to seemingly allow for easier detection. The DSSD detector is a hybrid SSD model made up of Residual-101 acting as the base network in replacement of VGG, and the original SSD network. These networks are combined together and augmented with deconvolutional layers (seen as the red layers on the extreme right of network in Figure 3.7) to help improve on the accuracy of detecting smaller objects in an image by increasing the resolution of feature map layers [Fu *et al.* 2017]. Also added are the Batch normalization layers which follow after the convolutions throughout the network. Additional information is passed from previous layers to the deconvolutional layers, therefore increasing the complexity of the network processes. SSD structures have been developed further with alternative base networks such as MobileNet, a network designed for mobile devices such as the Raspberry Pi or embedded vision applications. This model takes the form of factorised convolutions which factorizes standard convolutions into depth-wise and 1×1 convolutions with the aim of building light-weight deep neural networks [Howard *et al.* 2017]. Another model is the inception network, a combination of 1×1, 3×3, and 5×5 convolutional layers with their output filter banks concatenated into a single output vector forming the input of the next

stage of the pass through the network [Szegedy *et al.* 2015]. It performs dimensionality reduction wherever there is an increase in computational requirements.

3.3 You Only Look Once (YOLO)

YOLO is an object detection algorithm designed to be simple in its structure due to combining separate object detection components into one. It is also fast in comparison to other known object detection models during training and testing. It is made up of a fully convolutional network that takes a whole image as input to encode contextual information to do with the appearance of various object classes that the detector is trained on [Redmon *et al.* 2016]. Just like SSD, it treats the object detection task as a regression problem to learn class probabilities and bounding box coordinates [Soviany and Ionescu 2018].

The YOLO detection system follows a general three-step procedure as depicted in Figure 3.8. The steps are as follows:

1. Resizing images to 448×448 pixels for detection
2. Run a single convolutional network on the input image
3. Perform non-maximum surpression

3.3.1 YOLO structure

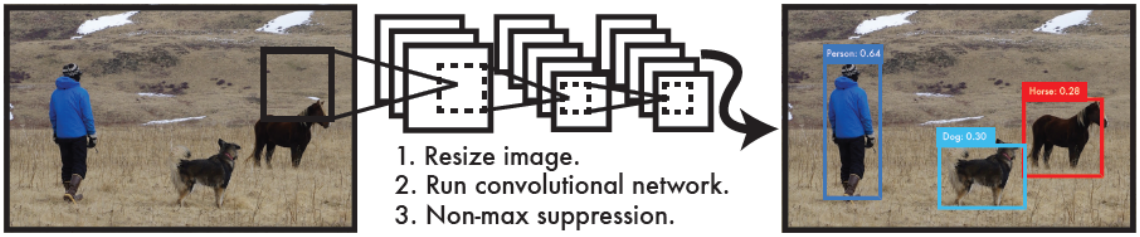


Figure 3.8: Sample input image fed into the YOLO algorithm for detecting a person, dog and cat. The image undergoes a generalised three-step process performed by the network which leads to detecting the target objects [Redmon *et al.* 2016]

The YOLO model is made up of a convolutional neural network. It starts off with a total of 24 convolutional layers followed by 2 fully connected layers as seen in Figure 3.9. It makes use of alternating 1×1 reduction layers to reduce the feature space from previous layers, which is then followed by 3×3 convolutional layers. To perform pretraining upon a dataset of images, the first 20 convolutional layers of the network are used. This is followed by the average-pooling layer and a fully connected layer. The last layer makes the class probabilities and bounding box coordinates predictions. The bounding box width and height are normalized by the width and height of the input image. The bounding box coordinates c_x and c_y are parametrized to be offsets of a specific cell location [Redmon *et al.* 2016]. All layers in the network, except the last layer, make use of the following linear activation function:

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases} \tag{3.7}$$

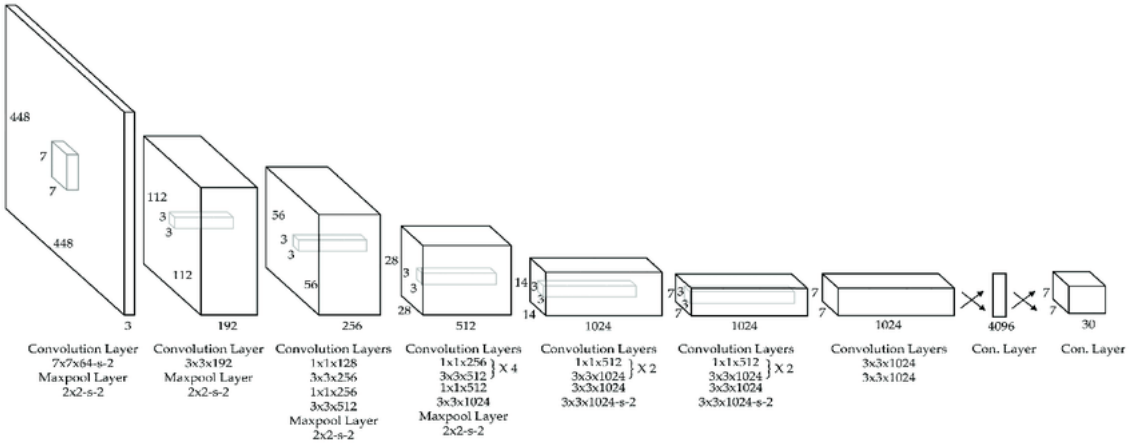


Figure 3.9: YOLO network design [Geethapriya *et al.* 2019]

There is a faster version of YOLO, referred to as *Fast YOLO*. The network is made up of 9 convolutional layers with fewer kernels than the original YOLO design.

3.3.2 Predictions using bounding boxes

Just as in SSD, the input images are divided into a grid of $N \times N$ cells. Within each cell, classification and localisation functions are applied to detect the target object(s). Each cell predicts B bounding boxes accompanied with confidence score S for each, and is given a label which the algorithm marks as a 1 where an object is present, and 0 otherwise. Geethapriya *et al.* [2019] represents the label L as a vector with multiple fields assigned to various features about the grid cell.

$$L = \begin{bmatrix} p_c & b_x & b_y & b_h & b_w & c_1 & c_2 \end{bmatrix}^T$$

Variable p_c takes the value 1 if a target object is within the cell or 0 otherwise. Variables b_x, b_y, b_w and b_h are the coordinates of the bounding box surrounding the target object if it is present, where b_x and b_y are the centre points of the box relative to the cell, while b_w and b_h are the width and height consecutively calculated with respect to the input image. Variables c_1 and c_2 represent two different classes of objects (more would be added in the case of having multiple object classes to detect). For example, if an object from class c_1 is contained within the cell, but not from class c_2, then $c_1 = 1$, while $c_2 = 0$. Only one cell is required to produce the bounding box containing the object. If more than one cell contains the same object, then the algorithm looks for the midpoint coordinates of the object. The cell containing this midpoint will be used instead of the other. In a scenario where multiple objects are present within a single image, it is possible to have midpoints of more than one object lying within the same cell. For two overlapping objects, the generated vector label L will contain the variables for both bounding boxes, therefore expanding the vector to contain 14 values, double the original size needed for one anchor box. Multiple bounding box predictors are produced within each grid cell, but only one predictor is required to make a final prediction per object, therefore NMS is applied. The bounding box with the highest IoU with the ground truth box is selected for predicting the sizes, aspect ratios, and class of objects [Redmon *et al.* 2016]. Each bounding box also has an accompanying

confidence score which demonstrates how confident the model is that an object is within the bounding box, calculated as:

$$S = Pr(Object) \times IoU_{pred}^{truth} \tag{3.8}$$

Since an input image is divided into a grid of $N \times N$ cells in which B bounding boxes are predicted in each with a total number of C class probabilities, and a bounding box produces 5 predictions (b_x, b_y, b_w, b_h, and S), then the output produced by the YOLO network is an $N \times N \times (B*5+C)$ tensor of predictions (Figure 3.3 shows a generalized tensor structure).

3.3.3 YOLO loss function

Upon the algorithm choosing the predicting bounding box with the highest IoU value, the YOLO loss function can then be applied. It is split into three parts which the algorithm implements, namely, the classification loss function, the localization loss function, and the confidence loss function each with unique purposes towards object detection.

Classification loss function

The Classification loss function is calculated for conditional class probabilities. For each cell within an $N \times N$ grid, the model predicts a set of class probabilities. If an object is found within a cell, the probability of the object belonging to every class C_i where $i = 1, 2, ..., n$ is predicted. The equation is as follows:

$$\sum_{i=0}^{s^2} \mathbb{1}_i^{obj} \sum_{c \in C_i} (p_i(c) - \hat{p}_i(c))^2 \tag{3.9}$$

In the formula above, variable $\mathbb{1}_i^{obj} = 1$ if the object is present in the cell, else $\mathbb{1}_i^{obj} = 0$. Variable $\hat{p}_i(c)$ is the conditional class probability for class $c \in C_i$.

Confidence loss function

The Confidence loss function is made up of two equations which are responsible for the "objectness" of the bounding box. If variable $\hat{C}_i$ represents the confidence score in grid cell i for bounding box j for when an object is detected, then the confidence loss is calculated with the following equation:

$$\sum_{i=0}^{s^2} \sum_{j=0}^{B} \mathbb{1}_{i,j}^{obj} (C_i - \hat{C}_i)^2 \tag{3.10}$$

If $\mathbb{1}_{i,j}^{noobj}$ is the complement of $\mathbb{1}_{i,j}^{obj}$, variable λ_{noobj} represents the weights when detecting background imagery, and no object is detected within bounding box j of grid cell i, then the confidence loss function equation is as follows:

$$\lambda_{noobj} \sum_{i=0}^{s^2} \sum_{j=0}^{B} \mathbb{1}_{i,j}^{noobj} (C_i - \hat{C}_i)^2 \tag{3.11}$$

Localisation loss function

The Localisation loss function measures the error between the ground truth and predicting bounding box in terms of size and location in the cell. If λ_{coord} influences the weight of the loss for predicted bounding box coordinates, variables x and y represent the bounding box centre points, while w and h represent the width and height of the bounding box, then the Localisation loss function can be formulated as follows:

$$\begin{aligned} &\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} \mathbb{1}_{i,j}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ &+ \lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} \mathbb{1}_{i,j}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \end{aligned} \tag{3.12}$$

Based on the paper authored by Redmon *et al.* [2016], combining all parts by calculating the sum of the classification, confidence, and localisation loss functions leads to the overall YOLO loss function formula:

$$\begin{aligned} &\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} \mathbb{1}_{i,j}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ &+ \lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} \mathbb{1}_{i,j}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\ &+ \sum_{i=0}^{s^2} \sum_{j=0}^{B} \mathbb{1}_{i,j}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{s^2} \sum_{j=0}^{B} \mathbb{1}_{i,j}^{noobj} (C_i - \hat{C}_i)^2 \\ &+ \sum_{i=0}^{s^2} \mathbb{1}_{i}^{obj} \sum_{c \in C_i} (p_i(c) - \hat{p}_i(c))^2 \end{aligned} \tag{3.13}$$

3.4 YOLOv2 (YOLO9000)

The original YOLO model still produces significant amounts of errors in areas such as localization and obtaining low recall predictions. In order to improve on the performance, Redmon and Farhadi [2017] developed an updated and 2nd model version of YOLO known as YOLO9000. Their paper title includes the words "*Better, Faster, Stronger*" to emphasize its enhanced capabilities. This real-time object detector has been pre-trained to detect over 9000 object types from the publicly available dataset named ImageNet [Deng *et al.* 2009]. This database is built from the WordNet database which places image categories into a semantic hierarchy structured as a tree of nouns that are related to one another. These connections are based on what the physical object is and how it conceptually connects to visual nouns. At the time of its release, it contained 12 subtrees with 5247 synsets and 3.2 million images [Deng *et al.* 2009]. From this, YOLOv2 developed and trained its localization and detection methods.

3.4.1 Error analysis of YOLO

The errors found in YOLO were deduced by comparing its performance with the two-stage object detector, Faster R-CNN. As a result, YOLO had two main areas in which improvements needed to be made. Firstly, YOLO did not perform as well in localization predictions. This translates into the predicting bounding box missing the target object on the input image. Secondly, YOLO obtained a low recall. Therefore, the focus in developing YOLOv2 is on improving the performance of localization and increasing the recall, while at the same time maintaining high classification accuracy.

3.4.2 Network design

VGG-16 is commonly used as the based network for single-stage object detectors. For YOLOv2, this base network has been replaced with Darknet-19. Darknet-19 is a convolutional neural network that contains 19 layers, hence its name. It also contains 5 max pooling layers. The base layer performs 5.58 billion operations for processing an input image over a single pass, as opposed to VGG-16 base network that performs much more operations with a count of 30.69 billion. This implies that the Darknet-19 model is a simplified base network. It also makes use of global average pooling to make its predictions. The 1×1 filters are applied to compress the feature representation between 3×3 convolution operations. All operations which Darknet-19 performs are detailed in Table 3.1.

The network has been modified for detection by replacing the last convolutional layer in the network with and additional three 3×3 convolutional layers each with 1024 filters. This is followed by a 1×1 layer with the required amount of outputs for detection. For fine grain feature use upon an image, a "passthrough" layer is placed between the last two convolutional layers. Batch normalization has been added to each convolutional layer in the YOLO network. It is used to regularize the model and eliminate dropouts from the model while at the same time avoiding overfitting. This helps to stabilize the training process and increase the rate at which convergence is reached. Consequently, the mAP improved by 2% [Redmon and Farhadi 2017].

3.4.3 Training

The first YOLO model trains on images with a resolution of 224×224 pixels. For the detection process, these images are resized to 448×448. The YOLOv2 network avoids having to simultaneously adjust to the resized objects and perform training by fine-tuning the network for 10 epochs to allow for filter adjustments of higher resolution images. This increased the mAP by 4%. Increasing the robustness of a detection model partially implies that the classifier can run on input images of different sizes. This has been included as an improvement from the original YOLO version into YOLOv2 by changing the input image resolution after every 10 batches. The dimension size is randomly selected from multiples of 32 ranging from 320 to 608. Therefore, an input image could be as small as 320×320 and as large as 608×608. These random changes in dimensionality enables the network to learn how to make predictions upon various input image sizes.

Type	Filters	Size	Output
Convolution	32	3×3	224×224
Maxpool	-	$2 \times 2/2$	112×112
Convolution	64	3×3	112×112
Maxpool	-	$2 \times 2/2$	56×57
Convolution	128	3×3	56×56
Convolution	64	1×1	56×56
Convolution	128	3×3	56×56
Maxpool	-	$2 \times 2/2$	28×28
Convolution	256	3×3	28×28
Convolution	128	1×1	28×28
Convolution	256	3×3	28×28
Maxpool	-	$2 \times 2/2$	14×14
Convolution	512	3×3	14×14
Convolution	256	1×1	14×14
Convolution	512	3×3	14×14
Convolution	256	1×1	14×14
Convolution	512	3×3	14×14
Maxpool	-	$2 \times 2/2$	7×7
Convolution	1024	3×3	7×7
Convolution	512	1×1	7×7
Convolution	1024	3×3	7×7
Convolution	512	1×1	7×7
Convolution	1024	3×3	7×7
Convolution	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Table 3.1: Darknet-19 convolution and maxpooling layer operations [Redmon and Farhadi 2017]

YOLOv2 uses a multi-label model of training which merges the classification and detection data training procedures. It makes use of labelled images based on object class to learn relevant information required for classification and making an accurate prediction using bounding boxes for objectness. During training, when the network picks up a labelled image for detection, backpropagation is performed based on the full YOLO loss function. When a classification image is picked up, backpropagation is performed only in parts of the architecture focused on classification.

Classification using WordNet

For the classification of objects in an input image, the WordNet tree structure is used to narrow down which object has been found. This is done by predicting conditional

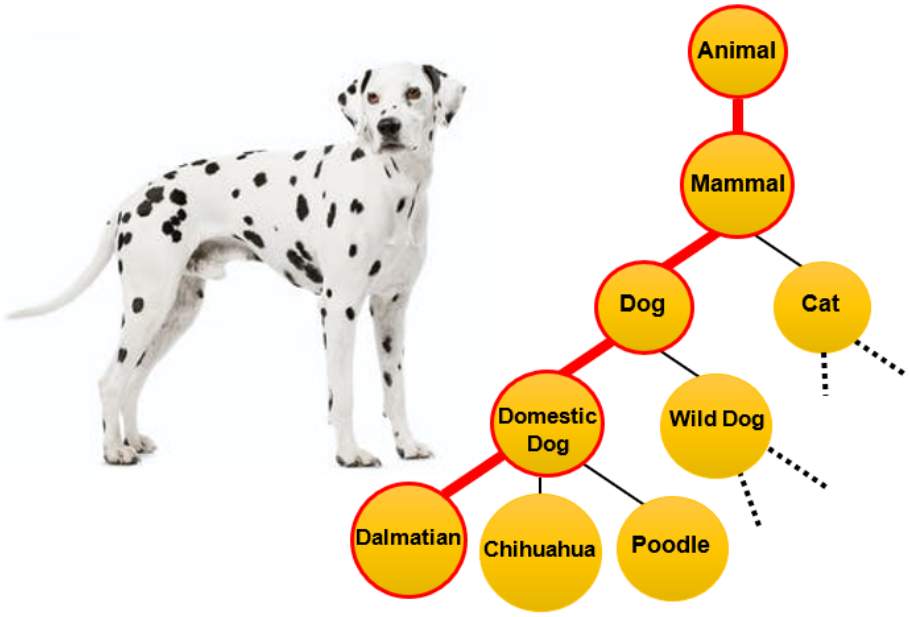


Figure 3.10: Subtree for classifying a Dalmatian

probabilities at each node of the tree. To calculate the probability of a certain node in the tree, the path from that node to the root is followed while multiplying all conditional probabilities. Figure 3.10 is an oversimplified example of how WordNet structures its subtrees and draws connections between nouns. Based on Figure 3.10, the probability for finding a Dalmatian within an input image is calculated as follows:

$$\begin{aligned} Pr(Dalmatian) = {} & Pr(Dalmatian|Domestic\ dog) \times Pr(Domestic\ dog|Dog) \\ & \times Pr(Dog|Mammal) \times Pr(Mammal|Animal) \\ & \times Pr(Animal|Physical\ object) \end{aligned} \tag{3.14}$$

An assumption is made that the input image contains a physical object, thus forming the probability $Pr(Physical\ object) = 1$. For detection, a bounding box is predicted along with the tree probabilities when traversing down the tree for the target object. The Highest confidence path is chosen between splits until a final path is mapped out to predict the object class. This method of classification is needed when combining multiple datasets that have varying naming conventions. For example, in the case of a dataset where firearms are sub-categorised further into pistols, shotguns, riffles, etc, the WordNet structure would have them linked up to the root node being a firearm. This allows for such a dataset to be combined with a binary labelled dataset where only the "FIREARM" and "NO FIREARM" classes exist.

3.4.4 Predictions using anchor boxes

In the original YOLO version, predicting the presence of an object is done using the IoU between the predicted bounding box and the ground truth box. Along with each of the 98 bounding boxes produced per image is a conditional probability for class predictions. This process still applies to YOLOv2, with a new addition of over 1000 anchor boxes. Object class and objectness are predicted for every anchor box produced. K-means clustering with

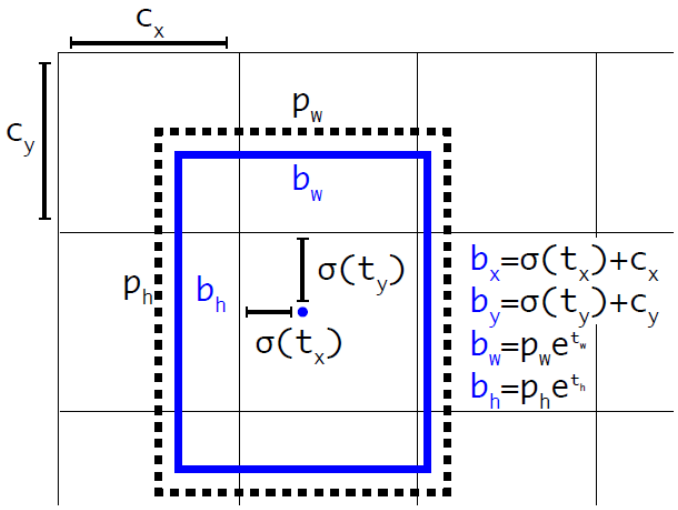


Figure 3.11: Bounding box location predictions [Redmon and Farhadi 2017]

different values of k is used to generate good priors for the dimensions of the bounding boxes. Priors that obtain high IoU values are considered to be good. This process is followed by plotting the average IoU with the closest cluster centroid. The distance metric used is as follows:

$$d(box, centroid) = 1 - IoU(box, centroid) \tag{3.15}$$

Coordinates are predicted relative to the location of the grid cells generated across the input image as portrayed in Figure 3.11. Logistic activation is used to bound the predictions between 0 and 1. Within each cell of the output feature map, 5 bounding boxes are predicted along with the values t_x, t_y, t_w, t_h, and t_0 for each [Redmon and Farhadi 2017]. If the cell coordinate at the top left corner of an image is represented by $[c_x, c_y]$, and the prior bounding box has a width and height represented by $[p_w, p_h]$, then the predictions for the bounding box are derived by the following:

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \\ Pr(object) &\times IoU(b, object) = \sigma(t_0) \end{aligned} \tag{3.16}$$

A passthrough layer has been added to YOLOv2 network which concatenates higher resolution features with the lower resolution. This transforms $26 \times 26 \times 512$ features maps to $13 \times 13 \times 2048$ dimensions by placing adjacent features into different channels. This allows for even small objects to be accurately located through having access to fine grained features. This process increases the performance of YOLO by 1%.

3.5 YOLOv3

With the development of algorithms such as SSD and RetinaNet, the accuracy of YOLOv2 became outperformed. This prompted the third version of YOLO to be released. It focuses on improving accuracy by using a hybrid underlying network made up of an integration of YOLOv2, Darknet and ResNet to form a new design structure for feature extraction [Jing *et al.* 2020]. Due to its complexity, this new architecture has a slightly decreased speed compared to YOLOv2.

3.5.1 Network design

Just as YOLOv2, Darknet still remains the base network of choice. However, it now contains 53 convolutional layers instead of the original 19 as with the previous version, thus its name becoming Darknet-53. For detection purposes, another 53 layers are included therefore creating a 106-layered structure. The full network architecture is seen in Figure 3.12 below. The convolutional layers are made up of 3×3 and 1×1 alternating filters. After each level of convolutions, batch normalization and dropouts are applied for training purposes. The multi-scale approach is used in the network to perform detection on different sized feature maps. This is done by applying 1×1 filters on feature maps of varying sizes placed in various parts of the networks. More specifically, detection is done at layer 79, 91, and 103 [Jing *et al.* 2020]. Semantic information gathered in images is increased due to the multi-layer convolution and pooling processes that operate throughout the network.

YOLOv3 has been further re-structured to detect objects ranging from large to small scales simultaneously. This is done using the Feature Pyramid Network (FPN) which is made up of scale-invariant image pyramids mapped to feature pyramids. Each scale computes features independently, therefore causing the shift of an object's scale up and down the pyramid levels [Lin *et al.* 2017]. This coincides with up-sampling used to obtain semantic features merged with low-level features - details of the target object and its location in the image based on information such as pixel intensity and pixel gradient orientation - to produce multi-scaled feature maps [Jing *et al.* 2020].

3.5.2 Class prediction

Just as with YOLOv2, the K-means clustering method is used for dimension clustering of anchor boxes on which objectness and class probability values are also predicted. A prediction vector is produced containing the coordinates of the bounding box, represented as b_x, b_y, b_w and b_h relative to the grid cells placed over the image, and a confidence threshold $\sigma(t_0)$ as an evaluation metric for scoring the model on detecting the object within each box.

The ground truth value $\hat{t}_*$ is calculated through the inversion of Equation (3.16). If t_* is the coordinate prediction value, then the gradient is calculated by finding the difference between the ground truth and predicted value as $\hat{t}_* - t_*$. Unlike YOLOv2, an objectness score is predicted for every predicted bounding box using logistic regression. The bounding box prior with the greatest overlap with a ground truth box is given a score of 1. All other predictions are ignored, therefore a one-to-one mapping is established between the chosen bounding box prior and the ground truth object.

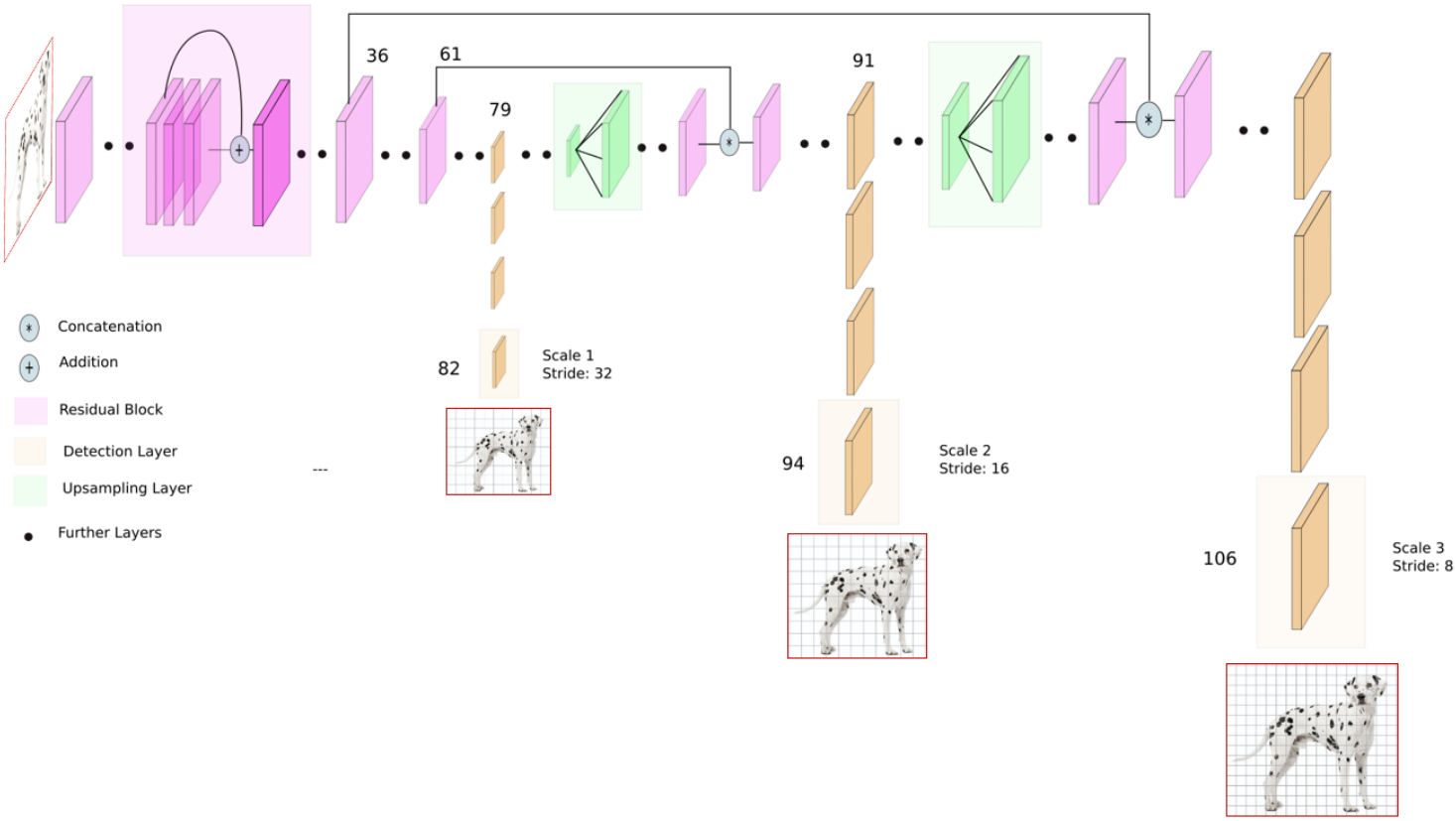


Figure 3.12: YOLOv3 Network Architecture [Jing *et al.* 2020]

For class predictions during training, the binary cross-entropy loss is used per predicted bounding box. YOLOv3 makes no use of softmax as it assumes that each bounding box has only one object class contained in it. In the case of a multi-class classification problem, this will most likely be a false assumption to make for all bounding boxes. Therefore, the multi-label approach is used to enhance the models data handling ability for multi-class purposes.

Multi-scale predictions

YOLOv3 makes predictions across three different scales in order to enhance the models ability to detect varying sizes of objects ranging from large to small objects in images. Within each scale, three anchor boxes are predicted, therefore producing a total number of 9 anchor boxes for this network. The scales are chosen through downsampling the dimensions of the input images by 32, 16, and 8 respectively. For instance, an input image of size 416×416 pixels would have feature maps with three different dimensions, including 13×13, 26×26, and, 52×52 on which target recognition is performed respectively. Convolutional layers are added onto the base feature extractor, of which the last of these added layers predicts a 3-d tensor [Redmon and Farhadi 2018]. This tensor encodes the three bounding box dimensions $N \times N$ at each scale, 4 bounding box offsets, one objectness prediction, and the k number of class predictions trained for the model to pick up. Therefore, the tensor is formulated as $N \times N \times [3 \cdot (4 + 1 + k)]$.

Feature maps from the previous 2 layers are upsampled by $\times 2$ dimensions. This is concatenated with feature maps earlier produced in the network in order to gather informative semantic features from the upsampled maps, and finer-grained features from the earlier maps. Convolutional layers are further added to process this newly concatenated feature map to predict a tensor upsampled by $\times 2$. This process is repeated for each scale up to the third scale, therefore inheriting all previous computation and feature identification to predict bounding boxes for objectness and classification.

3.6 Conclusion

This chapter looks at the design and architecture of different single stage object detectors in terms of their network structure, parameters used, and their methods of training and detection. The classifiers analysed include SSD, YOLOv1, YOLOv2, and YOLOv3.

SSD makes use of a convolutional network to compute feature maps and train on input images. It skips the region proposal stage and jumps to the processes of localization and object classification simultaneously in a single pass through the network. This helps to reduce the computational cost and increase the speed of training and inferencing, therefore making SSD a fast option. SSD is able to predict larger objects on images quite well, but has difficulty detecting small images during training due to poor extraction of features in shallow layers and feature loss in the deep layers [Shi *et al.* 2019].

All three versions of YOLO only need to go through an image in one pass (hence its name), therefore making the model an extremely fast detector. YOLOv1 splits the input image into an $S \times S$ grid of cells. Within each cell, the localization and classification process is applied simultaneously therefore giving each cell the ability to detect one object. However, this limits the maximum number of objects the model can detect. If there is more than one object present in a cell, the model will only be able to pick up on one of them. This is known as the *close object detection* problem, leading to more false negative cases and localization misses.

YOLOv2 has been designed to tackle the localization errors to garner better performance by adding batch normalization layers to the network. This improved the model's mAP performance by 2%. Secondly YOLOv2 is able to perform high resolution classification due to the fine tuning process that happens in the classification network. The resolution is increased to 448×448 before training, allow the network to adjust to the change. This increase the mAP by 4% [Kamal 2019]. The input image size changes to a randomly selected dimension after 10 batches during training allowing for objects to be detected at varying resolutions. Although YOLOv2 is an improvement to YOLOv1, it still has issues with accuracy and detecting small objects, and other models such as RetinaNet performed better.

YOLOv3 tackles the issue of detecting small objects by using short cut connections to obtain more detailed information from feature maps generated in previous layers. Its hybrid network structure consisting of Darknet-19 and the residual network produces 53 convolution layers, and the softmax layers have been removed entirely to allow for more

than one class of object to be found per grid cell. This is replaced with logistic classifiers which focus on individual training classes. These changes consequently improved on the model accuracy for class predictions. However, its performance decreased on larger objects in an image.

In this research, SSD inception v2 and YOLOv3 are the two single stage object detection models chosen for analysis and comparison with one another and with their counterpart two-stage object detectors. As a result of their state-of-the-art status and varying designs, they have been selected for the firearm detection task.

Chapter 4

Two-Stage Object Detectors

This chapter takes a look at the next generation of object detectors developed after the single stage models. Namely, two stage object detectors. In a similar manner to the previous chapter, an overview of two stage object detector structure is provided along with the individual procedures used to process an input image for training the classifier to perform detection.

Section 4.1 briefly outlines the difference between the single stage and two stage object detectors, with emphasis on the region proposal and classification stages. Section 4.2 discusses the Region Convolutional Neural Network (R-CNN) structure and processes from the feature extraction stage to testing. Section 4.3 discusses Fast R-CNN by summarising the enhancements made to R-CNN for its formulation. Section 4.4 presents Faster R-CNN by discussing its network structure and improvements made to the previous version. Section 4.5 presents Region-based Fully Convolutional Networks (RFCN) by looking at its architecture, training and inferencing processes.

4.1 Overview of two-stage object detectors

Single-stage detectors undergo the process of analysing input images by generating a series of predicted bounding boxes directly onto the area of interest within the image. A process of elimination through non-maximum suppression is conducted to isolate the bounding box that contains the most meaningful information in determining the object class, as well as its location as accurately as possible. An accurate prediction has the bounding box surrounding the target object in a perfectly fitted manner with its edges outlining the object in a rectangular form. In the case of two-stage object detectors, an accurate prediction is shown generally the same way as its counterpart. However, the process of reaching the goal of accurate classification and localisation is done in two stages, hence its name. This extended structure has made two-stage detectors to have high localization and object recognition accuracy [Jiao *et al.* 2019]. The first stage towards detection is the region proposal stage, and the second is the classification stage [Linh and Arai 2019], each with unique tasks. Figure 4.1 displays the generalised architecture that two-stage models are built on. It shows the connection between the early stage convolutional layers and the Region Proposal Network (RPN) that generates the Regions of Interest (RoI) to be fed into the classification network for RoI pooling and final classification and localisation predictions.

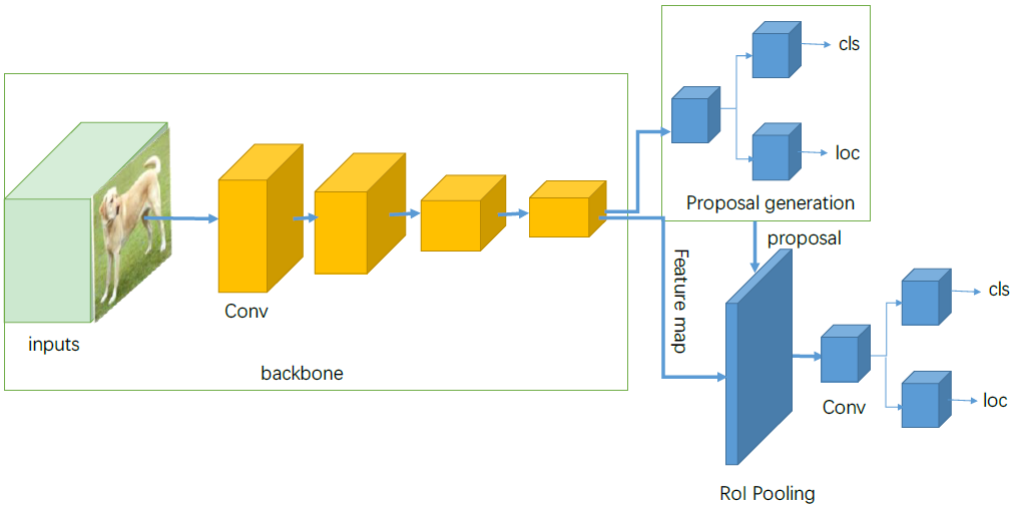


Figure 4.1: Two-stage object detector architecture. Inputs images are fed into a sequence of convolutional layers (*Conv*) represented by the yellow cubes. Feature maps are then generated and passed into the Region Proposal network. The blue cubes represent convolutional layers in this network from which classification (*cls*) and localisation (*loc*) are predicted.[Jiao *et al.* 2019]

Popular examples of two-stage object detectors include Mask R-CNN which presents its output in the form of segmentation masks, and Faster R-CNN which uses bounding boxes as outputs as done by the one-stage object detectors.

Region proposal stage

When an input image is fed into the network, this is the first stage that it encounters towards detection. Depending on the architectural design, various methods can be used to determine candidate regions of interest in which the target object is most likely to reside. Such examples include using the selective search algorithm (as done by Fast R-CNN) or RPN (as with Faster R-CNN). A classifier is used in this network to evaluate the probability of finding the target object within a suggested proposal region in which anchor boxes are predicted. The regression is performed on the coordinates of RoI.

Classification stage

The output in the form of proposed regions from the first stage are passed as input to the second stage of classification and bounding box regression for extraction of features. Bounding boxes are generated as possible locations for the target object for training of the algorithms which leads to the generation of a confidence score for the predicted class objects. Object localisation is determined using the IoU overlap between bounding boxes and ground truth boxes. A threshold is set for the overlaps to pin-point the boxes containing target objects.

4.2 Region Convolutional Neural Networks (R-CNN)

Combining of the Region Proposal method with CNNs has introduced the R-CNN model which proposes regions consisting of CNN features. This object detection system works on the localization problem by focusing its efforts within the proposed image regions for both object detection and semantic segmentation [Girshick *et al.* 2014]. In order to achieve the object detection task, the system has been segmented into 3 modules. The first is the actual generation of region proposals as candidate detections towards finding the target class/classes. The second module applies to every proposed region, in which a CNN is used to extract a feature vector of fixed length from each. Lastly, a category-specific linear Support VectMachine (SVM) is called to classify each region.

4.2.1 Feature extraction

The selective search algorithm is used for generating region proposals needed for detecting the target object. The process of feature extraction is bound and conducted within each proposed region, in which 4096-dimensional feature vectors are extracted using the Convolutional Architecture for Fast Feature Embedding (Caffe) framework. In order to extract features from these regions, the image data is scaled to a 227×227 pixel size for compatibility with the CNN structure. All pixels within the regions are warped into a tight bounding box for training which gives a look of the target object filling the bounding box. For testing, the selective search algorithm is implemented to extract approximately 2000 region proposals. Each region on the test image is warped and forward propagated through the neural network for computing feature vectors. SVM is used to score these extracted feature vectors for each target class. After which, non-maximum suppression is independently implemented for each target class with the task to ignore regions experiencing an IoU overlap with a higher scoring region surpassing the predefined threshold.

The computed dot products for an image in the network are performed for each class between features, SVM weights, and non-maximum suppression. They are placed into a single matrix product therefore producing a 2000×4096 feature matrix and a $4096 \times N$ SVM weight matrix, where N represents the number of target classes.

4.2.2 Training

Stochastic Gradient Descent (SGD) is a training procedure to adapt the CNN for detection using only warped region proposals. The regions which have an IoU overlap with a ground truth box ≥ 0.5 are set as positives for the appearing class, while the rest are set as negative (background region). Due to the higher number of occurrences for negative cases compared to positives, the learning rate is initially set to 0.001 for fine-tuning of $N + 1$ classes during each iteration of training, where N refers to the number of classes trained on, and the 1 extra class for background data. Furthermore, a window batch of size 128 is constructed at each SGD iteration comprising of 32 positive windows and 96 background windows which are uniformly sampled over all classes.

For training the linear SVMs for classification, the ground truth boxes are labelled as positive samples. A threshold overlap value of 0.3 is set to differentiate between types of proposed regions. Therefore, those which fall below 0.3 are classified as negative samples, while those which are above 0.3 but do not match the ground truth box are disregarded.

The SVM is then optimized for each class after all training labels have been set and features extracted.

4.2.3 Bounding box regression

After applying the class-specific SVMs to the regions found by the selective search, a new bounding box is predicted. It uses bounding box regression in an effort to improve localization in which there is a regression from features computed by CNN. The training algorithm is designed to map the ground truth box G to the proposed box P where $P^i = (P^i_x, P^i_y, P^i_w, P^i_h)$ represents the set of pixel-wise coordinates of proposal box P^i. The pixel-wise centre coordinates of proposal box P for training pair i is P^i_x and P^i_y, while the width and height of the proposal boxes at training pair i are represented by P^i_w and P^i_h respectively. The same principle applies for the ground truth box where $G = (G_x, G_y, G_w, G_h)$ representing the pixel-wise centre coordinates (G_x and G_y), as well as the width and height of the ground truth box (G_w and G_h). The input is then formed using P^i and G^i as a set of N training pairs as $\{(P^i, G^i)\}_{i=1,\ldots,N}$. The following transformations are applied to translate an input proposal P into ground-truth box G:

$$\begin{aligned}
\hat{G}_x &= P_w d_x(P) + P_x \\
\hat{G}_y &= P_h d_y(P) + P_y \\
\hat{G}_w &= P_w \exp(d_w(P)) \\
\hat{G}_h &= P_h \exp(d_h(P))
\end{aligned} \tag{4.1}$$

The transformations which map proposed box P to ground-truth box G are parametrized as functions of $d_*(P)$ where $*$ is x, y, w, h separately to form four functions. The functions $d_x(P)$ and $d_y(P)$ refer to a scale-invariant translation of bounding box P's centre point, while the functions $d_w(P)$ and $d_h(P)$ refer to log-space translations of bounding box P's width and height [Girshick *et al.* 2014]. For proposal P, a linear function $\phi_5(P)$ of the pool_5 features models each of these functions. The vector w_* consists of learnable model parameters which are used to form the function $d_*(P) = w_*^T \phi_5(P)$. Vector w_* is learnt by optimizing ridge regression using the following equation:

$$w_* = \underset{\hat{w}_*}{arg\,min} \sum_i^N (t^i_* - \hat{w}_*^T \phi_5(P^i))^2 + \lambda ||\hat{w}_*||^2 \tag{4.2}$$

In the paper written by Girshick *et al.* [2014], the value of $\lambda = 1000$ based on the validation set used during implementation of the model. Variable t_* represents the regression targets for the training pair (P, G) computed as:

$$\begin{aligned}
t_x &= (G_x - P_x)/P_w \\
t_y &= (G_y - P_y)/P_h \\
t_w &= \log(G_w/P_w) \\
t_h &= \log(G_h/P_h)
\end{aligned} \tag{4.3}$$

For the model to learn efficiently, it considers the proposal P that is near a ground truth box G. All other proposals further away from G are discarded. Closeness in proximity between P and G is determined by those which have a maximum IoU overlap greater than

a set threshold value (set at 0.6 for validation purposes). Each target class undergoes this process once for learning class-specific bounding-box regressors. For testing, each proposal is scored and a new window is then predicted.

4.3 Fast R-CNN

4.3.1 Limitations of R-CNN

For detection using the VGG16 network, the preceding R-CNN model takes 47 seconds to give an output on a Graphics Processing Unit (GPU). The GPU is a computing technology designed to accelerate and enhance graphical workload performance. Originally, it uses centred around real-time 3D graphical applications requiring high amounts of processing power (such as in gaming). Due to the acceleration abilities of a GPU on processing power, it has been adapted into high performance computing and deep learning applications. However, the rate at which R-CNN produces outputs with a GPU indicates that R-CNN is extremely slow and inadequate for real-time detection. The slowness is as a result of not distributing any computation when a convolutional network forward pass is performed for each object proposal. Along with the amount of time required is also the amount of space needed to perform optimally. The extracted features from each proposal in each image requires hundreds of gigabytes to be stored into a disk. Ideas such as making use of Spatial pyramid pooling networks (SPPnets) were implemented to share computation in an effort to speed up R-CNN by computing a feature map for whole input image from which feature vectors are extracted. Each object proposal is then classified using the extracted feature vectors. Max pooling is used in these proposal regions for feature extraction. Just as with spatial pyramid pooling, a number of output sizes are pooled and concatenated. As a result, feature extraction is performed faster, therefore allowing R-CNN to improve its speed at training and test time. However, this did not eliminate the amount of disk space needed nor did it decrease the tasks in the training pipeline made up of convolutional network fine tuning, SVM class-specific detection, and bounding box regressor learning. Lastly, the convolutional layers preceding the sparial pyramid pooling are never updated, therefore causing those layers to remain fixed. This has a negative impact on the accuracy of deep networks.

In order to fix these disadvantages associated with R-CNN and its SPPnets enhancement, the Fast R-CNN algorithm has been designed with higher speeds during training and testing, as well as better accuracy.

4.3.2 Architectural Design of Fast R-CNN

In Figure 4.2, the architecture behind the Fast R-CNN model is portrayed. Just as with R-CNN, it takes in a whole input image as well as multiple RoIs into the fully convolutional network. A feature map of a fixed-size is generated by passing the input into convolutional and max pooling layers. Afterwards comes the RoI pooling layer which uses the generated feature map to extract feature vectors. These vectors are passed into fully connected layers in the network which connect to two output layers at the end: the softmax probabilities which produces estimates for N object classes and the background class, and secondly the bounding-box regression offsets which produces four numbers indicating the bounding-box coordinates as an output for each object class represented during training.

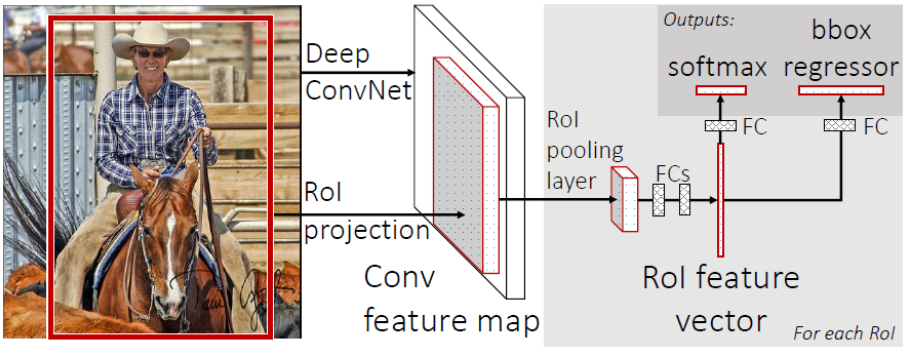


Figure 4.2: Fast R-CNN Architecture [Girshick 2015]

RoI pooling layer

Every RoI produced is represented by the position it falls within the convolutional feature map as a four-tuple (r, c, h, w). Coordinates (r, c) refer to the top-left corner of the RoI window, while coordinates (h, w) are the height and width respectively. Max pooling takes place in the RoI pooling layer (see Figure 4.2) which converts the features lying within the RoIs into $H \times W$ sized feature maps. The paper written by [Girshick 2015] explains that RoI max pooling operates by taking an RoI window of size $h\ timesw$, and dividing it into an $H \times W$ grid of sub-windows. These sub-windows are approximately $h/H \times w/W$ in size each. Within these sub-windows are values which undergo max-pooling into the corresponding grid cell.

During training, a total of N images are sampled in Stochastic Gradient Descent (SGD) mini-batches of size R. At a later stage R/N RoIs undergo sampling from each image. Let $N = 1$ and $x_i \in \mathbb{R}$ be the i^{th} activation input into the RoI pooling layer while the j^{th} output from the r^{th} RoI found in the image is represented as y_{rj}. In addition, let $R(r, j)$ be the set of input indexes within the area in which y_{rj} undergoes maxpooling. If $i(r, j) = argmax_{i' \in R(r,j)} x_{i'}$ is the index set of inputs, then the RoI pooling layer evaluates the output unit $y_{rj} = x_{i'*(r,j)}$ as the max pools in the sub-window. Partial derivatives are routed into the RoI pooling layer via back-propagation. These derivatives of the loss function are computed in this layer as:

$$\frac{\partial L}{\partial x_i} = \sum_r \sum_j [i = i^*(r, j)] \frac{\partial L}{\partial y_{r,j}} \tag{4.4}$$

4.3.3 Fine-tuning for training

Fast R-CNN makes use of a fine-tuning process during training which concurrently configures the bounding-box regressors and softmax classifiers for the fully connected layers in the network. These layers using zero-mean Gaussian distributions undergo an initialisation using standard deviations 0.01 for softmax classification and 0.001 for bounding-box regression. Each layer in the network uses 3 different learning rates: per-layer learning rate set at 1, the learning rate for biases at 2, and the global learning rate set at 0.001. The learning rate decreases as the training progresses. Research conducted by [Girshick 2015] compares the use of the softmax classifier to linear SVMs (as used by the R-CNN model) to establish the better choice in terms of mAP using the PASCAL VOC 2007 trainval

dataset. The results are described in Table 4.1. Three pre-trained ImageNet models are used to perform the comparison, including the CaffeNet which is referred to as model "S" (small), the wider network VGG_CNN_M_1024 named "M" (medium), and lastly VGG16 network being the largest model thus refered to as "L" (large). For all three models, softmax outperforms SVM, which indicates that the fine-tuning process of Fast R-CNN in one combination has better results than the multi-stage pipeline approach previously used.

Classifier	**S**	**M**	**L**
SVM	56.3	58.7	66.8
Softmax	57.1	59.2	66.9

Table 4.1: Softmax vs SVM in Fast R-CNN network [Girshick 2015]

The Fast R-CNN network has two output layers. The first releases an output per RoI which is produced using softmax. The output is in the form of a probability distribution described as $p = (p_0, ...p_K)$ representing discrete values for $K + 1$ classes. The second layer releases bounding-box regression offsets for all K object classes under training as an output of the form $t^k = (t^k_x, t^k_y, t^k_w, t^k_h)$, where k represents each class, while t^k is the scale-invariant translation and log-space horizontal and vertical shifts with respect to the proposal [Girshick 2015]. Labels are applied to the RoIs during training, where u specifies the ground-truth class, while v specifies the ground-truth bounding box regression target. To train on these RoIs for bounding-box regression and classification, two log loss functions are equated together. Firstly, the log loss for the u class defined as $L_{cls}(p, u) = -\log p_u$. Secondly, there is the log loss for bounding box regression targets for class u, where $v = (v_x, v_y, v_w, v_h,)$, and $t^u = (t^u_x, t^u_y, t^u_w, t^u_h)$ for class u predictions. Therefore, the bounding box regression log loss is defined as:

$$\begin{aligned} L_{loc}(t^u, v) &= \sum_{i \in x,y,w,h} smooth_{L_1}(t^u_i - v_i) \\ \text{where} \quad smooth_{L_1}(x) &= \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| \ - \ 0.5 & \text{otherwise} \end{cases} \end{aligned} \tag{4.5}$$

Finally, the loss function used for bounding box regression and classification jointly is thus:

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t^u, v) \tag{4.6}$$

The hyper-parameter λ is placed to control the balance between the two equated loss functions

Another process which occurs during fine-tuning is the SGD mini-batch sampling from randomly selected images for training. These images go through data augmentation in which they are flipped horizontally with a probability of 0.5. A total of 128 mini-batches are constructed, and 68 RoIs are sampled per image. The RoIs are separated into two groups. One contains 25% of these RoIs which meet the criteria of having an IoU overlap ≥ 0.5 with the ground-truth box. They contain labels with a foreground object class where $u \geq 1$. The second group contains the rest of the RoIs which have an IoU with the ground-truth box in the value range $0.1 \leq$ IoU < 0.5, referring to background information labelled as $u = 0$.

4.3.4 Faster detection

Once all object proposals have been defined and the network has undergone fine-tuning, the detection stage follows which involves taking a list of images and $R \approx 2000$ object proposals to score. During testing, a forward pass is run to output a class probability distribution p for each RoI r, including predicted bounding-box offsets for all K classes in the network. For each object class k, RoI r is assigned a confidence value using the probability $\Pr(\text{class} = k|r) \triangleq p_k$. Lastly, non-maximum suppression is applied for each class to narrow down the number of bounding boxes.

The speed to compute fully connected layers in the network for whole-image classification is increased by compressing the layers with truncated Singular Value Decomposition (SVD) which performs dimensionality reduction on the input images [Pedregosa *et al.* 2011]. In order to accelerate the detection process, the truncated SVD is added to the Fast R-CNN network to speed up the forward pass of computing the fully connected layers [Jiao *et al.* 2019]. Given the matrices U, $\sum_t$, and V, where U represents the $u \times t$ matrix made up of t left-singular vectors of the weight matrix W, $\sum_t$ represents the diagonal $t \times t$ matrix for all top t singular values of W, and V is a $v \times t$ matrix made up of the first t right-singular vectors of W, then the layer within the network parametrized by W using SVD is computed as:

$$W \approx U \sum\nolimits_t V^T \tag{4.7}$$

Compressing a network helps to boost the speed of processing especially with a large number of RoIs. This compression in Fast R-CNN works by taking two fully connected layers used by matrices $\sum_t V^T$ and U respectively, to replace the single fully connected layer that corresponds with matrix W.

4.4 Faster R-CNN

Though Fast R-CNN shows improvement around accurate detection of objects compared to RCNN, it still suffers from slow computation. This is as a result of the time taken during detection using selective search. It requires generating 2000 proposals for each input image, which impacts the training time and therefore slowing down the detection process [Saxena 2020]. In an effort to move closer to designing a model with faster processes leading up to real-time detection, the state-of-the-art Faster R-CNN model has been designed. It has a similar network structure as the predecessor (Fast R-CNN) comprising of two modules. The first consisting of fully convolutional network, and the second is the Region Proposal Network which makes use of proposed Regions, all of which are combined to form a single object detection system as depicted in Figure 4.3.

4.4.1 Convolutional layers

When an input image is first fed into the network, it passes through convolutional layers built on top of a base network such as the VGG-16 architecture (as is the case with SSD and Faster R-CNN models). At this stage, the extraction of features matching the target objects is done by a sliding window (or filter) undergoing training to pick up on useful data that will assist with object detection. This process of extraction by the sliding window leads to the construction of feature maps [Khazri 2019]. The feature maps in turn capture

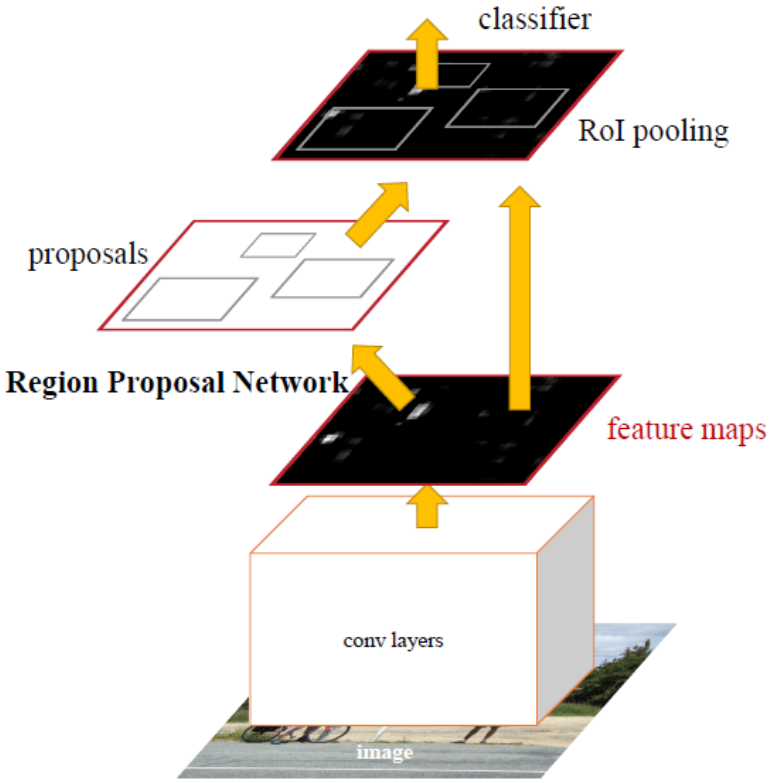


Figure 4.3: Faster R-CNN network consisting of convolutional layers and the RPN [Ren *et al.* 2015a]

the results of applying the sliding window on input images [Brownlee 2019]. Varying feature details are stored and represented in the different feature maps produced.

4.4.2 Anchor boxes

Anchor boxes are used for each predictor to find the most probable location, shape and size of the object it is trained to detect [Christiansen 2018]. At each step of the sliding window gliding across the input images, anchors are generated. Thus, if a feature map is of the dimension $W \times H$, then the total number of anchor boxes produced is $W \times H \times k$, where k is the total number of region proposals generated at specific sliding window locations within an input image. This directly implies that the regression (reg) layer generates $4k$ outputs relating to the anchor box coordinates, while the classification (cls) layer has $2k$ outputs which are probability score estimates for the presence or absence of target objects.

An architecture is loosely said to have the translation invariant property if its output is less sensitive to translations of the input [Kauderer-Abrams 2017], in which the object contained in the image is translated in various possible forms. It also plays a role in reducing the model size [Ren *et al.* 2015a]. Faster R-CNN adopts this property in terms of anchors and proposal regions. When the proposal goes through a translation, the network is able to still correctly predict the proposal in any location. The network also makes use of a multi-sclae design based on anchors. Just as in Fast R-CNN, single-scale feature maps

and images are enough to derive and compute convolutional features. Along with the sliding window running across the feature maps as a filter, a pyramid formation of anchors is used which is computational efficient to classify and regress bounding boxes using anchor boxes of multiple scales. Its design allows for features to be shared across without incurring scaling compute costs.

4.4.3 Region Proposal Network (RPN)

The architecture is designed to be fed input images of various sizes. A set of region proposals are generated for each image all of which have an objectness score referring to the confidence of finding a target object within the proposed rectangular region. The generating process involves overlapping a small-sized network with the convolutional feature map output. The network is then fed an $n \times n$ spatial window that is mapped to a lower dimensional feature from the convolutional feature map. This feature in-turn becomes the input for the box-regressor and box-classification fully connected layers which are shared throughout the network in spatial locations due to the sliding window approach operations of the network.

Training of RPNs

Back propagation and SGD are both used during the training process. The losses derived from the loss function are back-propagated and SGD mini-batches are constructed from each input image consisting of both positive anchors which match with the ground truth and negative anchors which fall below the threshold.

A total of 256 anchors are randomly sampled in each image to avoid having a bias for negative sampled anchors when optimizing the loss function for anchor boxes. This occurs due to the negative samples being more dominant in the set. The loss function of each mini-batch is evaluated based on the chosen samples where there exists a 1:1 ratio between the positive and negative samples. New layers are initialised using weights from a zero-mean Gaussian distribution with standard deviation 0.01 [Ren *et al.* 2015a]. The shared convolutional layers are also initialised and fine-tuned before the iteration of training.

Loss Function

A binary class label is firstly assigned to the generated anchors which reflect objectness. A positive label refers to a detected object. This is dependent on the IoU overlap between an anchor and ground truth box. The anchor with the highest IoU or an IoU overlap with the ground truth ≥ 0.7 are assigned positive labels. If the IoU overlap is found to be < 0.3, then a negative label is assigned. It is possible for positive labels to be assigned to more than one anchor at once by a single ground-truth box. In such scenarios, NMS is applied to narrow down the numbers by removing the lower scoring boxes. Based on these criteria [Ren *et al.* 2015a], the loss function can be defined for an image:

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \tag{4.8}$$

Reflected in the loss function are variables $\{p_i\}\}$ and $\{t_i\}$ hold the outputs for the classification and regression layers respectively. Variable p_i represents the probability for a

predicted anchor box i, t_i is a vector containing the 4 coordinates for anchor boxes while t_i^* is a vector containing the same type of information for the ground truth box. Variable p_i^* is the label assigned to the ground truth boxes which hold the value 1 if the anchor box is positive and 0 if negative. The λ symbol is a balancing parameter set as $\lambda = 10$, while N_{cls} and N_{reg} are the mini-batch size and number of anchor boxes consecutively which normalize the two terms in the equation [Ren *et al.* 2015a]. Classification loss L_{cls} and regression loss L_{reg} are both used in the loss function equation. The log loss across both label classes are reflected in L_{cls}, while $L_{reg}(t_i, t_i^*) = R(t_i - t_i^*)$ where R is the function $smooth_{L_1}$ from Equation (4.5) used in Fast R-CNN's regression loss function. Variable $p_i^* = 1$ in the term $p_i^* L_{reg}$ when the regression loss activates for positive anchors, and for the negative case the regression loss is deactivated making $p_i^* = 0$.

Bounding box regression

Parametrization of the bounding box coordinates is handled below as regression from an anchor box to the neighbouring ground truth box:

$$\begin{aligned} t_x &= x - x_a/w_a \ , \ t_y = y - y_a/h_a \\ t_w &= \log w/w_a \ , \ t_h = \log h/h_a \\ t_x^* &= x^* - x_a/w_a \ , \ t_y^* = y^* - y_a/h_a \\ t_w^* &= \log w^*/w_a \ , \ t_h^* = \log h^*/h_a \end{aligned} \tag{4.9}$$

Coordinates x, y are the centre points of each predicted bounding box, while w, h represent the width and height. Values x_a, y_a, w_a and h_a represent the same features for the anchor boxes, as do values x^*, y^*, w^* and h^* for the ground truth boxes.

The feature maps generated contain regression features all of spatial size 3×3. In order to handle various sizes of predicted bounding boxes, n bounding box regressors are learned by the algorithm with each managing their own scale and aspect ratio and without the sharing of weights.

4.4.4 RoI pooling and detection

Before an actual detection can be made, Faster R-CNN applies RoI pooling on the features maps for each region proposal produced in the RPN. The size of RoI pooling is dependent on the number region proposal that are sectioned, which helps to improve processing speeds at this stage. The feature maps are reduced in size and sent to fully connected layers for the classification stage. By using two different fully connected layers, a prediction is made for each target object [Grel 2017].

4.5 Region-based Fully Convolutional Networks (RFCN)

4.5.1 RFCN Architecture

The RFCN architecture makes use of the two-stage object detection design which starts with the region proposal network stage, followed by region classification. As opposed to Fast/Faster R-CNN, this network is fully convolutional, shares computation on the whole image [Dai *et al.* 2016], and uses 101-layer Residual Net (ResNet-101) as the base network. When an input image is fed into the network, it passes through the 100 convolutional layers

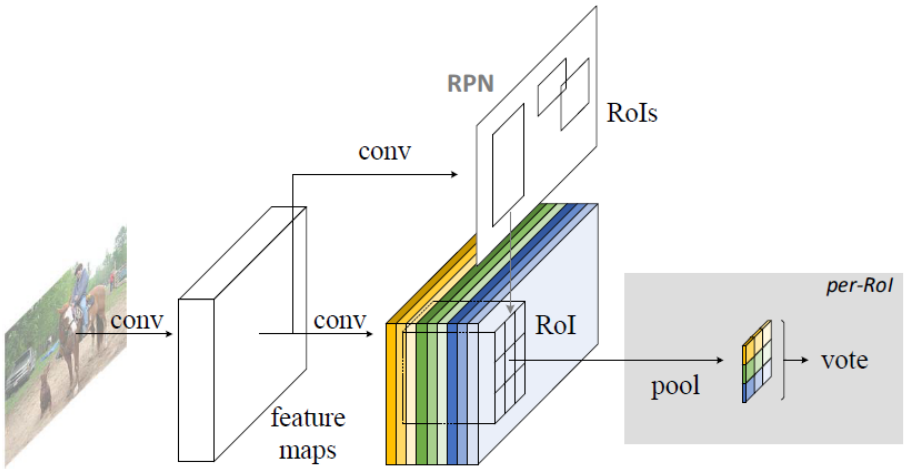


Figure 4.4: Architecture of RFCN from end-to-end [Dai *et al.* 2016]

which form Resnet-101. The last layer of ResNet-101 is a 2048 dimensional convolutional layer attached to an initialised 1024 dimensional 1×1 convolutional layer for reducing the input image dimension and releasing an output of feature maps which are passed to the RPN for RoI generation. Each RoI is classified into one out of $C + 1$ object classes (with an extra 1 representing background image information).

Translation variance has been brought into the FCN design to assist with the localisation task, done by using a set of k^2 position-sensitive score maps encoded with location data highlighting the object spatial location as an output for specialized convolutional layers. Following this structure is the RoI pooling layer that directs the score map outputs. It is placed between sets of convolutional layers shared on the whole image to create an RoI-wise subnetwork to increase accuracy of detection. Figure 4.4 depicts the overall architecture showing the flow of information from the convolutional layers into the RPN where RoIs are created and applied to score maps.

4.5.2 Position-sensitive score maps & Position-sensitive RoI

The last convolutional layer produces a $k^2(C + 1)$-channel output layer formed from the $C + 1$ categories and the set of k^2 score maps, which results in the production of k^2 score maps for each class object [Dai *et al.* 2016]. The RoIs are split into a $k \times k$ grid where each bin (cell) contains information corresponding with object spatial location. Therefore, if $k = 3$, the number of score maps produced is $k^2 = 3 \times 3 = 9$ each encoded with specific cases related to grid location such as: Top Left, Top Centre, Top Right,..., Bottom Right. Furthermore, if the RoI is of the dimension $w \times h$ (corresponding to width and height), then each cell is approximately of the size $\frac{w}{k} \times \frac{h}{k}$. If the tuple (x_0, y_0) represents the top-left corner of and RoI, $z_{i,j,c}$ defines one of the score maps generated, Θ represents every parameter that is learnt in the network, while n denotes the total amount of pixels per cell, then the position-sensitive RoI average pooling operation is set in each (i, j) cell which pools score map (i, j) as follows:

$$r_c(i, j|\Theta) = \sum_{(x,y)\in \text{bin}(i,j)} z_{i,j,c}(x + x_0, y + y_0|\Theta)/n \; ; \quad \text{where } i \geq 0, j \leq k - 1 \tag{4.10}$$

The pooling output $r_c(i,j)$ is that of cell (i,j) for category c. A voting process takes place on the RoI which is done by averaging the scores to produce the $(C+1)$-dimensional vector per RoI. The k^2 generated score maps together produce a total of k^2 position sensitive scores which are involved in voting of the RoI. This is done using the following summation operation:

$$r_c(\Theta) = \sum_{i,j} r_c(i,j|\Theta) \tag{4.11}$$

The softmax is applied afterwards to the $(C+1)$-dimensional vector for all class objects to assist during training with finding the cross-entropy loss, and also during inferencing to rank the RoIs. The softmax operation is defined by the following:

$$s_c(\Theta) = e^{r_c(\Theta)} / \sum_{c'=0}^{C} e^{r_{c'}(\Theta)} \tag{4.12}$$

Bounding box regression

To apply bounding box regression, convolutional filters are used to produce $4k^2$ maps. Position-sensitive RoI pooling is applied onto these maps to compute a $4k^2$-dimensional vector per RoI, where each element holds a boundary box [Hui 2018]. The average of the vector values is calculated using average voting which creates a newly formed 4-dimensional vector that defines bounding boxes using the following parametrization: $t = (t_x, t_y, t_w, t_h,)$

4.5.3 Training and Inferencing

The speed of computation during training and inferencing is increased as a result of not having learnable layers following the RoI layer. However, an RoI loss function is still applied. During training, RoIs are assigned positive labels if their overlap with the ground truth box produces an IoU ≥ 0.5. In contrast, IoU overlap values which fall below that threshold are assigned negative labels.

Loss Function

If the cross-entropy loss for classification is defined as $L_{cls}(s_{c^*}) = -\log(s_{c^*})$ where c^* is the label applied for ground-truth RoIs, and the loss for bounding box regression L_{reg} is the same loss as what is used by Girshick [2015] for Fast R-CNN, then the RoI loss function for RFCN is defined by combining both regression and classification losses together as follows:

$$L(s, t_{x,y,w,h}) = L_{cls}(s_{c^*}) + \lambda[c^* > 0] L_{reg}(t, t^*) \tag{4.13}$$

In this loss equation, t^* represents the ground truth bounding boxes, while λ is used as a weight balancer.

Online Hard Example Mining (OHEM)

The OHEM algorithm has been designed for training detectors that are region-based in nature which selects hard positive and negative examples for training [Shrivastava *et al.* 2016]. The loss function is computed for each RoI formed to determine the network's performance on them via a forward pass in the network. Hard examples are choosen from the group of generated RoIs by sorting them based on computed loss. From N proposals,

a total number of B RoIs with the highest losses are selected as those which the model performed worst on, to form B/N examples on which Back propagation is implemented. An IoU threshold value is set at 0.7 which NMS uses to remove RoIs that have lower computed losses but high overlaps with selected regions. This eliminates de-duplications of RoIs in the same location with co-related losses.

Activation score voting

A comparison evaluation is done on 300 RoIs during inferencing, which are generated from the RPN portion and are given scores per training category. The output undergoes NMS using 0.3 as the IoU threshold value to select the most fitting regions. Finally, the voting process is conducted to determine whether or not a target object has been detected. The position-sensitive maps have varying levels of activation in different locations of the image depending on the probability of detecting an object. A bin is strongly activated in specific positions therefore attaining a high score if a portion of the object is contained in it. If most of the bins are activated in this manner, the voting process will produce high scores. If however, there is an overlap that is slightly off with the ground truth box, then the bins which do not land on the target object will not be activated, hence leading to a vote of lower scores. The scores produced from the maps are averaged out using softmax to classify the object leading to a prediction being made.

4.6 Conclusion

This chapter takes a deep dive into the different two stage object detector architectures based on their network structure, method of training and making predictions for classification and localisation. The architectures discussed include R-CNN, Fast R-CNN, Faster R-CNN, and RFCN.

The R-CNN model combines CNNs with the region proposal method, which takes care of the localization problem by using the selective search algorithm to generate 2000 regions spread across the image to perform classification and localization within. The workload is then distributed between these region proposals allowing for the classifiers to focus on their specific areas. However, no form of learning takes place when the selective search algorithm is implemented which can result in bad candidate region proposals. The model is also computationally heavy compared to the single stage detectors, and therefore training the network takes alot of time. Inferencing can take up to as long as 47 seconds which is not ideal for real-time detections [Gandhi 2018].

Fast R-CNN had been designed to speed up the processes of its predecessor by feeding the CNN with input images rather than region proposals. This generate convolutional feature maps from which the region proposals are extracted. This eliminates the need to feed 2000 region proposals into the CNN, therefore speeding up the training time. However, the fact that region proposals are generated using the selective search algorithm still slows down the computation and forms a bottleneck during training.

Faster R-CNN has similar processes with Fast R-CNN, but instead of using the drawn out processes of the selective algorithm, Faster R-CNN has another independent network dedicated for predicting region proposals, called the region proposal network which takes

care of the bottleneck effect that occurred in Fast R-CNN. It is significantly more accurate than its predecessors because of the introduction of the RoI pooling layers used to classify objects within the region proposals. The draw back is that the model is computationally heavy, which in effect means that training time is still very long.

RFCN focuses on distributing the region proposal computational load within the network by having convolutional learnable layers applied on the whole image, therefore speeding up the computation process. It is also a translation invariant model, meaning that it is able to detect an object correctly regardless of the objects shifting locations on the image. The model takes after Faster R-CNN by using RoIs to extract features, therefore obtaining a higher detection accuracy compared to Fast R-CNN. However, it makes use of residual networks to balance out accuracy and speed. This causes a trade-off, making the model to be faster than Faster R-CNN during training and inferencing (the computational load reduction also impacted its speed) but less accurate.

The chosen two stage object detection models to be analysed and compared are RFCN Resnet 101 and Faster R-CNN ResNet 101 (both consisting of 101 deep layers) because they are state-of-the-art models in the deep learning space. Both models were created after Fast/Faster R-CNN in an effort to outperform previous versions of two-stage object detectors. However, Faster R-CNN and RFCN have different architectural designs, therefore their performances can be compared with one another.

Chapter 5

Research Methodology

Present security surveillance systems still require a huge amount of human intervention to activate the alert. During a criminal attack, the victim's reaction is dependent on their reflexive manner of stress handling. Research shows that victims are usually caught off-guard, therefore causing them to physically respond out of fear using the "fight-or-flight" approach. Such reflexive reactions cause a time delay in contacting the necessary security personnel, which can result in receiving a late security response. The proposed model for firearm detection does not involve any form of human intervention for recognition. Instead, the model works by automatically identifying firearms in images. The Research Methodology provides a full description of the methods used in this research to capture the performance of each classifier.

Section 5.1 lists the research questions. Section 5.2 discuses the aims and objectives to be met by this research. Section 5.3 motivates the reasoning behind performing firearm detection research. Section 5.4 contains the methods used to carry out the research, which includes dataset collection, preprocessing procedure, and the training and testing stages.

5.1 Research question

1. What are the state-of-the-art deep learning object detection methods that can be used for vision-based firearm detection?
2. Which deep learning method performs fastest for detection of firearms on images?
3. What is the best performing deep learning method in terms of high accuracy for firearm detection based on its overall F_1 score?

5.2 Aim and Objectives

The aim of this research is to test the performances of the state-of-the-art deep learning-based object detection methods in the firearm detection task. This can be done by ranking the classifiers using two separate metrics. Firstly, based on the speed of inferencing during detection, and secondly on the level of accuracy based on each method's F_1 score calculation. In order to achieve this aim, we set the following objectives:

- Identify the deep learning based object detection methods that can be used for vision-based firearm detection.

- Collect a dataset in which the deep learning classifiers can train and test on for firearm detection.
- Train and test the deep learning classifiers with the collected dataset of images.
- Compare and contrast the performance of the classifiers for firearm detection based on detection accuracy and inference speed.

5.3 Research method

5.3.1 Dataset

The dataset has been divided into two classes. Namely, the positive class (firearm), and the negative class (not a firearm). Data collection has been done to fit this binary format. A total of 3400 images have been collected with an 80-20 split between the train and test set. The dataset deliberately consists of firearm images in various orientations, different lighting conditions and environments. This is done to allow for models to train from as many scenarios as possible in which a firearm could be found. It is taken from four public domains:

Soft Computing and Intelligent Information Systems (SCI^{2}S)

The SCI^{2}S database [Triguero *et al.* 2017] consists of 3000 images from the "region of proposal" subset. For the purpose of this research, a total of 1937 images were extracted. The selection process is based on image quality and relevance. Images which are cartoon-based or extremely blurry were not selected since the intention is to recognise firearms in realistic scenarios.

Internet Movie Firearms Database (IMFD)

The IMFD database [IMFDB 2007] contains images from a randomized selection of action movies. Each image is a scene which contains at least one firearm. A total of 538 images has been taken from this database. Images taken align with the two most preferred firearms used for criminal offences in South Africa, handguns and .38 specials (i.e. pistols and revolvers). Videos within this database promoting responsible gun usage has also been extracted into frames to formulate an additional 100 images.

Kaggle

Kaggle is an online platform in which machine learning and data science engineers are able to find and publish datasets for their research purposes. It offers free GPU cloud environments for training deep learning models. A total of 732 images have been collected from the Handgun Detection dataset [Larxel 2020] within kaggle.

Google

Pictures of drills and hair dryers have been taken from Google Images. These objects have deliberately been selected in order to differentiate between firearms and objects that can easily be mistaken for a firearm based on its shape and manner of use. These are merged with cropped out images of firearms to train on to formulate an additinal 93 images.

5.3.2 Preprocessing

The preprocessing procedure involves resizing of images followed by annotating each image by labelling the target object. The images collected have been resized to dimensions no greater than 1024×768 in order to increase the speed of training time per classifier. The labels are manually placed in the form of a bounding box surrounding the firearm. Everything outside the bounding box forms part of the negative class of objects. The *labelImg* software [Tzutalin 2015] is used for this labelling task of the whole dataset.

Pascal Visual Object Classes (PASCAL-VOC)

Labelling of images is done in PASCAL-VOC format for all the classifiers (except YOLO) to make use of during training. In this format, the bounding box coordinates x_min, x_max, y_min, y_max are stored in an xml file to be used for locating the firearm position within each image. This information is saved under the "GUN" class.

YOLO

YOLO requires its annotations to be stored in the YOLO format due to its architectural design. Annotations per image are placed within a .txt file consisting of the object class (0 represents the firearm class), x_centre and y_centre coordinates, as well as the width and height of each bounding box.

5.3.3 Deep learning classifiers and training

Effectiveness of training is largely dependent on the relevance of data collected, thoroughness applied to preprocessing technique, amount of data, computing power, and the type of classifier used. The proposed approach for this research towards training classifiers is the region of proposal method. For firearm recognition, this involves finding all possible positions in an image that a firearm may more likely be located. This probability is based on the labelled location of firearms in the training set. The classifiers that will be used for training a firearm detection model are all deep learning algorithms, briefly detailed below:

1. **Single Shot Multibox Detector (SSD inception version 2)**:
   An object detection architecture designed for real-time purposes. It works by extracting feature maps, and then applying convolutional filters for detection. SSD is fast with inferencing as a result of eliminating the region proposal network, and instead uses multiboxes.

2. **You Only Look Once (YOLO version 3)**:
   A multi-object detection algorithm which feeds an input image into a single convolutional network. It uses the IoU (Intersect over Union) and NMS (Non-maximum suppression) processes to predict the object and its location within an image.

3. **Region-Based Fully Convolutional Network (RFCN ResNet101)**:
   RFCN is an object classifier that takes in feature maps and applies convolutional filters to create position-sensitive score maps. The position-sensitive RoI pool is applied to generate a multidimensional vote array. The array is then averaged out and uses softmax for object classification.

4. **Faster Region-Convolutional Neural Network (Faster R-CNN ResNet101)**: The Faster R-CNN architecture makes use of a region proposal network design that forms bounding boxes and utilizes them to classify objects.

K-fold cross validation

Studies done by Vabalas *et al.* [2019] show that small datasets trained for classification purposes tend to achieve a higher accuracy than larger datasets due to overfitting. Tests by Vabalas *et al.* [2019] were performed upon datasets containing 200 to 1000 samples. A common approach used to counter this challenge in computer vision is the K-fold approach, in which the dataset is split between train and validation sets K times in order to get a circulation of different train and validation samples. The dataset size used in this research is considered to be relatively small. Hence, this method has been adapted in which $K = 5$. Each classifier has undergone training 5 times using different variations of the train and validation sets. The average of these results is computed to output the classifier's final performance.

5.3.4 Testing, Evaluation, and Optimisation

Trained models for firearm detection are evaluated on the test images. Each model attempts to solve the binary classification problem by classifying the test samples into either the FIREARM class or NO FIREARM class. In order to evaluate the performance of trained models, the F_1 score, a statistical measure used commonly for classification problems which makes use of precision (p) and recall (r), is used with the assumption that labels of each test image is already known. Precision measures the proportion of positive predictions from the test set that were correctly identified out of all positive predictions, as follows:

$$p = \frac{TP}{TP + FP} \text{ ; } \textit{where } \text{TP} = \text{True Positive, FP} = \text{False Positive}$$

Recall measures the proportion of actual positives in the test set that is correctly identified by the trained model from the pool of known positives, as follows:

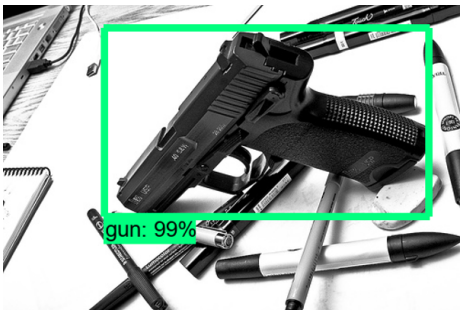
$$r = \frac{TP}{TP + FN} \text{ ; } \textit{where } \text{TP} = \text{True Positive, FN} = \text{False Negative}$$

Finally, the F_1 score is a function of both precision and recall to compute an accuracy measure for the trained model's results, as follows:

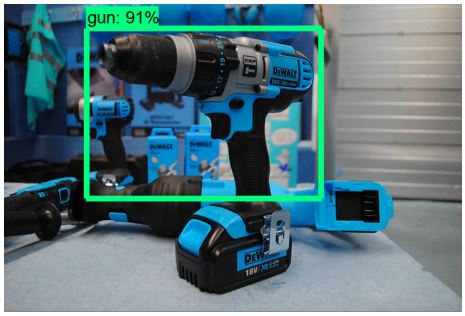
$$F_1 = 2 \times \frac{p \times r}{p + r} \text{ ; } \textit{where } 0 \leq F_1 \leq 1$$

The confusion matrix below visually portrays the predictions made by each trained model upon the test set, where positive predictions indicate the presence of a firearm while negative predictions indicate that there is no firearm. The subscript *pred* represents the predictions made by the classifier, while subscript *true* represents the true class.

	POSITIVE $_{pred}$	**NEGATIVE** $_{pred}$
POSITIVE $_{true}$	TP	FN
NEGATIVE $_{true}$	FP	TN



(a) TP output



(b) FP output



(c) TN output



(d) FN output

Figure 5.1: Examples of possible detection outputs

Differentiation between FP, TP, FN, TN

Both classification and localisation predictions have equal influence over the type of prediction made by the trained models. Since the gun class is the only object that has been trained on, a bounding box surrounding the predicted gun object will appear during inferencing with the "gun" label attached. The following scenarios, also displayed in Figure 5.1, determine what type of prediction has been made:

1. If the predicting bounding box overlaps the ground truth, the prediction overall for that input is classified as **TP** (see Figure 5.1a).
2. When a gun prediction is made with a bounding box not overlapping the ground truth object (whether or not the gun is present in the image), this prediction is considered a **FP** (see Figure 5.1b).
3. If the model does not make a classification or localisation prediction upon an image without a gun, this prediction is considered a **TN** (see Figure 5.1c).
4. If a gun is present in the image, but is not picked up by the trained model, this prediction is considered a **FN** (see Figure 5.1d).

5.4 Conclusion

A detailed description of the methodology applied for this research has been laid out in terms of the research question, aim, and motivation behind the proposed methodology.

Procedures towards implementation and evaluation metrics of trained models has also been discussed in full and backed up with preliminary results.

Chapter 6

Experimental results and discussions

6.1 Introduction

In this chapter, the performance of each classifier is analysed in detail with statistics in terms of accuracy results produced, as well as the average speed of processing during detection. The best performing classifiers are ranked based on three separate metrics. Namely, the highest accuracy obtained using the F_1 score, as well as the speed at which each classifier was able to analyse each image during inferencing. The Loss results have also been detailed for each classifier as a reflection of their performance during training.

6.1.1 Training software

All classifiers were trained using python scripts in which the deep layers were configured. The code and the dataset will be shared later once the copyright related issues are resolved. The Tensorflow API [Abadi *et al.* 2015] is a public open-source interface that offers tools to build, train, and test deep learning models. This API was used to train SSD, RFCN, and Faster R-CNN. During training, Tensorflow version 1.15 which includes GPU support for CUDA-enabled cards had been used. Using this platform allowed for each classifier to be trained for approximately 3-5 hours at a time. The ideal loss values are those which tend as close to 0 as possible. This reflects minimum error of the classifier for its localisation and classification predictions. Therefore, training stopped either when the classification and localisation loss decrease began to approach a constant value, or when both losses dipped down to less than 1 and remained consistent over time.

YOLO has been trained using the Darknet framework on Google Colab [Bisong 2019]. Darknet assists in loading weights and configuration files required to train, classify, and display the results obtained from the custom object detection model. The average training time of YOLO is the same as the training time for those trained on Tensorflow. Execution ended when the loss dipped down to 0.5.

6.2 Analysis of performance per classifier

A total of 3400 labelled images were fed into each classifier, in which 2720 were set aside as the training set, and 680 as the validation set. The time it takes to train each classifier is highly dependent on the processing power of the devices used as well as the computational heaviness of each classifiers architecture. Each classifier went through training 5 times on 5 varying training sets. They each have an associated validation set which are unique to each training set. These training and validation pairs have been labelled as Group A, B, C, D, and E. The validation set for one group, is fed in as training for another group, therefore allowing for the mixture of different images in the training sets. To produce the validation set, the 3400 images have been divided into 5 equal portions, where each portion is paired with a training set which does not consist of the 680 images set aside for validation. The results obtained for each group will be combined together for an overall classifier performance analysis.

6.2.1 F_1 score evaluation

The evaluation metric used for each group per classifier is the F_1 score as a function of precision and recall. The number of True positives and negatives, as well as False positives and negatives have been counted to generate the final precision, recall and F_1 score. A separate unseen test set of 202 images containing 0 or more firearms each, have been put together to evaluate the trained classifiers in this manner. This accuracy metric is commonly used to compare the performance of classifiers trained on the same dataset for multi-class predictions [Grüne 2020]. Possible F_1 scores range between 0 and 1 where 0 indicates that there were no correct predictions, while 1 indicates that all predictions were correct. The F_1 scores have been converted into percentages for each group per classifier, and are ranked from the highest value which is associated with the best performing group down to the lowest performing group with the lowest F_1 score. The final F_1 score for each classifier is the average of the attained group scores.

6.2.2 Classification, Localisation, and Total Loss

The loss metric used to measure the difference between the true class and prediction class is tracked during the training process of each classifier. This is generated using the loss function associated with each classifier during its forward pass. The results obtained from the loss function during the course of training are generated from evaluating the trained model at that point in time on the validation set. The loss eventually appears to be negative-logarithmic in nature if trained for long enough. It depicts how well the trained model is performing at each stage of training upon the validation set by summing up the errors made for each example. The expected trend during training is for the error to gradually decrease after each iteration until it averages out and remains within a certain range with marginal change over time. Lower loss/error values indicate a better trained model, ideally nearing as close to 0 as possible. Therefore, the best performing model is associated with a value closest to 0. There are three loss metrics that have been evaluated:

1. **Classification Loss**: This loss metric is used for discrete predictions such as the firearm detection problem. It shows the performance of classifying objects within the dataset by making use of the ground truth classification as a reference. Smaller values for classification loss indicate less error in misclassification of an object.

2. **Localisation Loss**: The localisation loss is a smooth L1 loss between the predicted bounding box correction and the ground truth values [Weng 2018]. Localisation considers the IoU value between the prediction and ground truth box. This overlap is constantly evaluated by using the $t = 0.5$ threshold value where values greater than t indicate a bigger overlap with the ground-truth bounding box. The IoU and localisation loss have an inverse relationship. The higher the IoU value, the smaller the localisation loss becomes, as it reflects the amount of error in determining the object location.

3. **Total Loss**: The total loss combines all losses together to produce an overall loss for the classifier. This includes the classification, localisation, regularisation, and clone losses.

6.2.3 Inference speed evaluation

The speed of computation during testing is evaluated for each model. This is done to determine which classifiers are best matched to performing detections in real-time speeds, which is measured to a couple of milliseconds. The timer is set to begin from when the inference processing initiates, and stops when the prediction is made by the classifier. The average inference speed is computed for each classifier by timing the detections per image, and passing those values into the mean formula to find the average for that classifier. This final average represents the speed attained for that specific classifier.

The hardware configuration of the device used for testing the trained classifiers SSD, Faster R-CNN, and RFCN consists of an Intel Core i5 CPU with a processing speed of 1.6GHz, and 8GB of RAM space. The YOLO classifier was tested directly on a Google Colab virtual machine. It runs on an Nvidia K80 GPU processing unit with a processing speed 0.82GHz , and 12GB of memory.

6.2.4 SSD Observations

Precision, recall, and F_1 score

The SSD model has been trained on the 5 different groups of training sets. Detailed in Table 6.1 are the results of the trained classifiers upon each group on the test set generated for this evaluation phase. Each result is associated with the group which the classifier used for training and validation. Henceforth, these classifiers will be called Group A, B, C, D, and E to emphasize the training and validation set that was used specifically for that version of the classifier. The TP, TP, TN, and FN values shown in table 6.1 details the number of predictions found in the test set. All classifier groups performed firearm detection considerably well on the test set. Group B obtained the highest F_1 score at 85% giving it the best firearm detection performance in the group, while Group C obtained the lowest F_1 score at 71%. In order to obtain realistic statistics to overcome the overfitting problem, the average values have been computed for precision, recall, and the F_1 score.

Group	TP	FP	TN	FN	Precision	Recall	F_1 score
A	89	16	105	22	0.85	0.80	0.82
B	88	12	109	18	0.88	0.83	0.85
C	65	8	111	44	0.89	0.60	0.71
D	77	17	101	33	0.82	0.70	0.75
E	69	11	108	39	0.86	0.64	0.73
				Average:	**0.86**	**0.89**	**0.77**

Table 6.1: Precision, Recall, and F_1 score results for SSD using K-fold validation

The results show that SSD has an average precision of 86%, recall at 89%, and the F_1 score at an overall 77%. For all 5 groups, the recall value is lower than precision. The difference ranges between 0.05 and 0.3. This indicates that the SSD model trained on firearm detection has a high rate of correct positive predictions out of the total pool of positive predictions made during testing. However, its recall value indicates that it is less able to pick up on true positives known from the pool of ground truths. On average, there is a 20% more chance of getting a FN detection than a FP detection. Figure 6.1 shows a sample of detections made by the trained SSD model.



Figure 6.1: SSD firearm detection samples from Group B classifier results. SSD is easily able to detect firearms that contrast with background colours. The top-right and bottom-left images each contain a firearm which are small in size and are coloured similarly to its background. This creates a blending effect which caused the SSD classifier to predict a FN.

Loss values

During training, the SSD model produced the classification, localisation, and total loss values which are a reflection of the performance of each model upon the validation set. Table 6.2 details the resulting loss values on the different validation sets at the end of training:

Group	Classification loss	Localisation Loss	Total Loss
A	0.80	0.20	1.65
B	0.95	0.16	1.60
C	1.65	0.37	2.70
D	1.20	0.19	1.78
E	1.50	0.22	2.20
Average	**1.22**	**0.23**	**1.78**

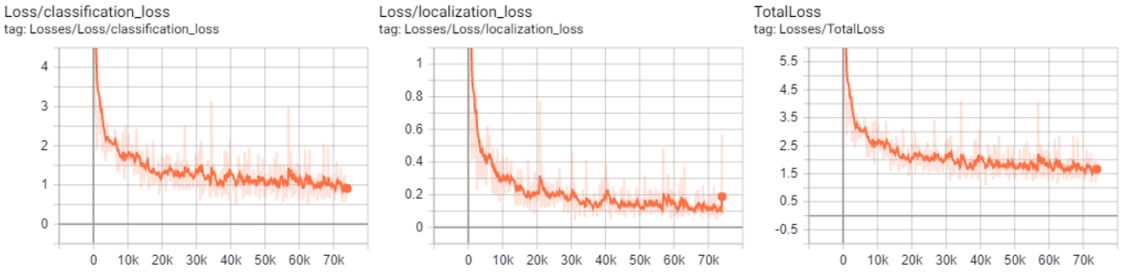
Table 6.2: Loss values obtained for SSD trained model

For all groups, the localisation loss is measured less than the classification loss. The localisation losses depicted here are all less than 0.5, indicating that the model has been trained long enough to correctly locate a gun when predicted. However, the classification loss values range between 0.8 and 1.65, which shows that the summed errors are slightly higher than that for the localisation loss, which ties together with the precision and recall results obtained. In terms of visual firearm recognition, this means that the model was more easily able to accurately detect the "no gun" class as opposed to the "gun" class. This confirms the higher FN values in comparison to the FP values.

Group A achieved the lowest classification loss, giving it the highest classification performance. This also shows in the high number of TP detections it achieved. The highest classification loss was achieved by Group C at 1.65. It achieved the worst performance in classification, indicating that it was less able to detect firearms than its counterpart SSD models. This can be seen in the number of TP values obtained in Table 6.1 where it obtained the lowest value.

The lowest Total Loss value in the group is that achieved by Group B, while the highest is achieved by Group C. This means that Group B overall obtained the smallest error when detecting, while Groups C obtained the highest summed errors. The average Total Loss for SSD is 1.78, while the average classification and localisation loss respectively are 1.22 and 0.23.

During training, the loss values obtained after a couple of thousand iterations was recorded. There was a significant increase in performance of classification and localisation achieved as the models continued to undergo the training process, as shown in Figure 6.2.



(a) SSD Group A Loss Graphs (from left to right) Classification loss, Localisation loss, and Total loss

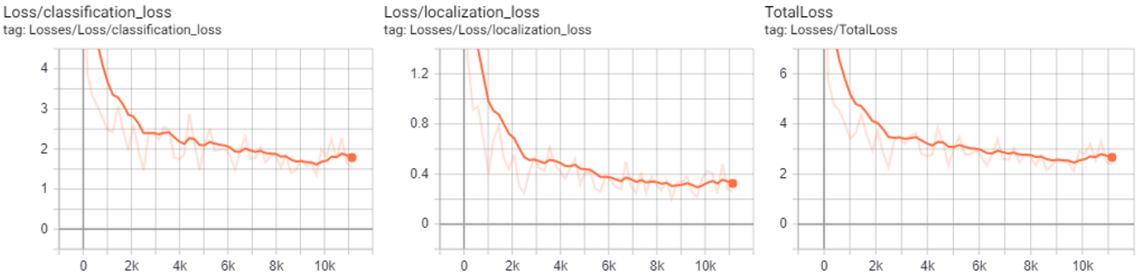
Figure 6.2: Loss graphs produced during training for SSD



(b) SSD Group B Loss Graphs (from left to right) Classification loss, Localisation loss, and Total loss



(c) SSD Group C Loss Graphs (from left to right) Classification loss, Localisation loss, and Total loss



(d) SSD Group D Loss Graphs (from left to right) Classification loss, Localisation loss, and Total loss



(e) SSD Group E Loss Graphs (from left to right) Classification loss, Localisation loss, and Total loss

Figure 6.2: Loss graphs produced during training for SSD

Inferencing Speed

SSD has been designed to work for real-time detection, therefore making it able to perform processing fast enough to handle real-time conditions. Unlike other classifiers, SSD speeds up processing by not making use of region proposal networks. The average inferencing

speed of SSD is approximately 38.6 milliseconds. The inferencing speeds obtained for each group is displayed in Table 6.3.

Group	Inferencing Speed (millisec)
A	39
B	38
C	39
D	39
E	38
Average	**38.6**

Table 6.3: Inferencing speed of SSD

SSD analysis

The average F_1 score of the SSD model is 77%. Based on the images that were tested upon, the SSD model struggled to identify firearms in which the background colour matches closely with the firearm itself. This caused the firearm to blend into the image, therefore causing the SSD models to miss the firearm completely by associating it with the "no-gun" class. However, with contrasting backgrounds, the firearm is easily detected. SSD was able to detect firearms that were partially occluded, held in the hand, or placed openly on a surface.

The average localisation, classification, and total loss values of the trained SSD model are 0.23, 1.22, and 1.78 respectively. The model is able to accurately determine the location of the firearm in an image when the gun class is predicted, therefore reducing the number of FP detections. However the higher classification loss showed in the higher number of FN detections.

The average inference speed of SSD is 38.6 milliseconds. There is no significant difference in the inferencing speed between the different groups, indicating that the dataset had no influence on the time it took to detect. This is possibly linked to the design of SSD which makes no use of region proposals. As a result, the localisation task is able to implement within milliseconds.

6.2.5 YOLO Observations

Precision, recall, and F_1 score

YOLOv3 has been used for this research, and it predicts a confidence score for each bounding box using logistic regression. Its COCO mAP value range is between 51 to 61 measured at 0.5 [Redmon and Farhadi 2018]. Detailed in Table 6.4 is the precision, recall and F_1 score results based on the TP, FP, TN, FN detections obtained during testing.

Group	TP	FP	TN	FN	Precision	Recall	F_1 score
A	78	18	103	28	0.81	0.74	0.77
B	75	16	105	32	0.82	0.70	0.76
C	75	25	107	16	0.75	0.82	0.79
D	62	13	105	46	0.83	0.57	0.68
E	73	32	94	23	0.70	0.76	0.73
				Average:	**0.78**	**0.72**	**0.75**

Table 6.4: Precision, Recall, and F_1 score results for YOLO using K-fold validation

Group D obtained the highest precision value at 83%, while Group E obtained the lowest at 70%. This indicates that out of the pool of gun predictions made by the trained models, Group D had the most accurate detections, while Group E had the least. The highest recall value was obtained by Group C at 82%, while the lowest value of 57% was obtained by Group D, indicating that the model trained on Group C managed to accurately pick up on the most ground truths in the gun class, while Group D performed significantly less in this task. Furthermore, the highest F_1 score at 77% was obtained by Group A, and the lowest at 68% by Group D. However, for the sake of obtaining reasonable accuracies, the averages have been considered for the overall performance of YOLO. The average value of the precision, recall, and the F_1 score for YOLO is 78%, 72%, and 75% consecutively. This stipulates that YOLO has a higher proportion of accurate gun class predictions within the ground truth pool, therefore decreasing the chances of having an FN detection. However, this model has a 6.8% chance of picking up an FN detection over an FP detection. Figure 6.3 displays samples of the YOLO detections.



Figure 6.3: YOLO firearm detection samples taken from Group C results. Similar to SSD visual results, YOLO was mostly able to correctly classify and locate the firearm except where the firearm colour blended into the background as seen in the top-right and bottom-left images.

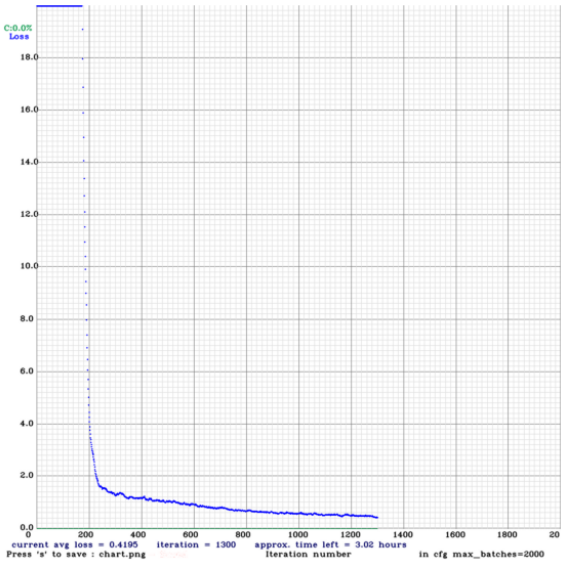
Loss values

The darknet framework combines the classification and localisation loss values by computing the overall loss upon each batch size of 64 during training. Therefore, the output produced is a loss that measures the overall performance of the trained YOLO model. Table 6.5 references the performance of all 5 trained YOLO models upon the validation set:

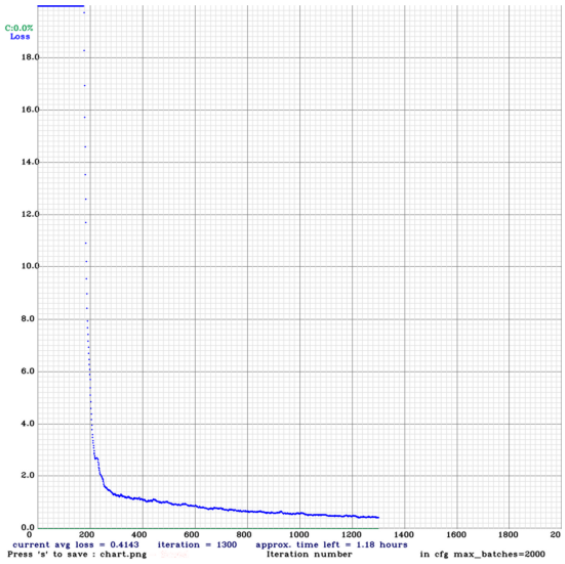
Group	Overall loss
A	0.4195
B	0.4871
C	0.4143
D	0.4751
E	0.4875
average	**0.4567**

Table 6.5: YOLO overall loss

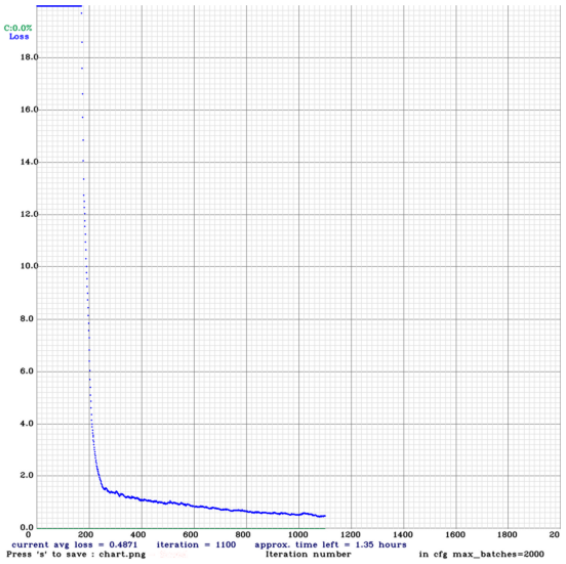
The closer the overall loss tends to 0, the higher the chances of obtaining good firearm detection performances based on both classification and localisation. The lowest overall loss was obtained by the model trained on Group C, achieving a score of 0.4143. This indicates that Group C has the lowest error rate, therefore giving it the best performance in the group overall which aligns with the F_1 score results. The lowest performing model, Group E, scored the highest overall loss at 0.4875. This does not align with the group that attained the lowest F_1 score, however it did obtain the lowest precision result. The recall score of the model trained on Group D was significantly lower than all other results, hence pushing down the F_1 score to give it the lowest rank. This translates into a minuscule loss difference of 0.04 between Group D and E models. The average overall loss obtained for YOLO is 0.4567.



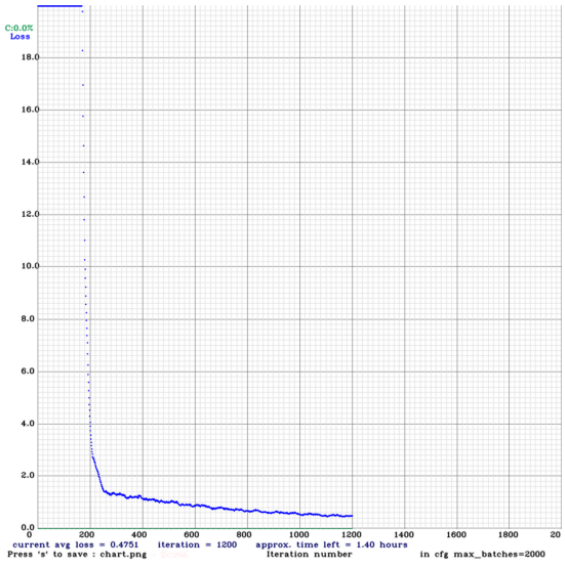
(a) YOLO Group A Loss Graphs



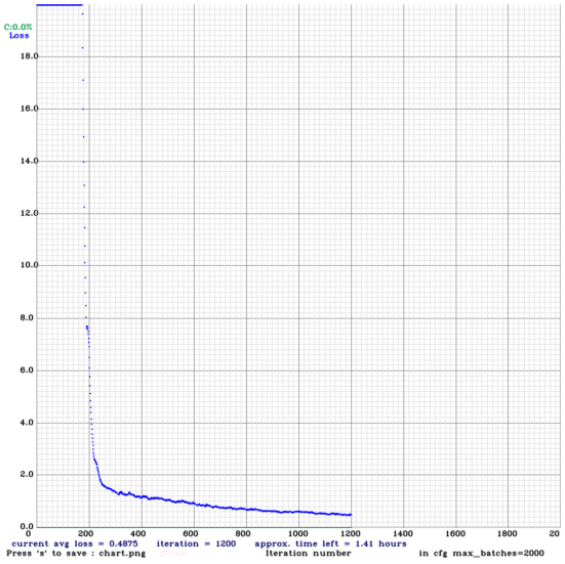
(b) YOLO Group B Loss Graphs



(c) YOLO Group C Loss Graphs



(d) YOLO Group D Loss Graphs



(e) YOLO Group E Loss Graphs

Figure 6.4: The YOLO loss graphs describe the change in average loss values over time during the training process. The expected output is a downward sloping graph which starts off with the loss dropping quickly in the first few hundred iterations. The graph begins to average out as the average loss value tends closer to 0. The current average loss is depicted on the y-axis, while the iteration/steps are shown on the x-axis. The blue symbols beneath each graph represent the average loss at the end of training (current avg loss), the last iteration before cut-off (iteration), and time left for training which is optionally pre-defined (approx. time left).

Inferencing Speed

YOLO is designed to process faster than region proposal models by making use of a single neural network. It takes in a whole image as its input to process directly, rather than using multiple proposed regions. As a result, the average processing time from the point of initiating the detection to the prediction display is 40.176 milliseconds. The inferencing speed achieved for each group can be found in Table 6.6.

Group	Inferencing Speed (millisec)
A	40.17
B	40.19
C	40.16
D	40.19
E	40.17
Average	**40.18**

Table 6.6: Inferencing speed of YOLO

YOLO Analysis

The F_1 score achieved by YOLO is relatively stable at 75%. All training processes terminated their execution after achieving a consistent loss value around the 0.5 mark. Over a period of time, it was clear that the loss value was not going to change much more than what it was already computing, hence the reason for termination. However, this seems to have had an influence on the detection. If trained for longer, YOLO may have achieved better results. In addition to this, the YOLO model seemed to have also struggled with detection where the background colour matches closely to that of the firearm.

The average overall loss obtained by YOLO is 0.4567. This aligns with the darknet framework training requirements in which a properly trained model needs to achieve an overall loss that is consistently below 0.5 over a period of time. The number of FP detections were less than half the FN detections, indicating that a small error was achieved for localisation, therefore YOLO is able to correctly identify where the firearm object is within the image when detected. However, it had a higher number of FN detections, therefore the classification loss needed further reduction. To counter this, the model needs to be trained for a longer period of time to reduce the overall error, which in turn will reduce the localisation and classification errors to a level that will positively impact how well it predicts detections.

The average inferencing speed for YOLO is 40.176 milliseconds. All inferencing speed times used to compute the average are relatively similar with differences no greater than 0.02 milliseconds. Therefore, the datasets composition had no influence on the time taken to inference. YOLO has been designed for inferencing at near real-time speeds, which it has managed to achieve for both localisation and classification.

6.2.6 RFCN Observations

Precision, recall, and F_1 score

The chosen RFCN resnet 101 classifier is publicly recorded to have a COCO mAP of 30 [Rathod and Shi], and displays its prediction in the form of a bounding box around the target object. K-Fold cross validation has been applied to produce 5 sets of results for precision, recall, and the F_1 score. The results are displayed in Table 6.7.

Group	TP	FP	TN	FN	Precision	Recall	F_1 score
A	100	36	85	10	0.74	0.91	0.81
B	98	30	91	9	0.77	0.92	0.83
C	96	40	82	10	0.71	0.91	0.79
D	106	32	93	11	0.77	0.91	0.83
E	96	45	80	11	0.68	0.90	0.77
				Average:	**0.73**	**0.91**	**0.81**

Table 6.7: Precision, Recall, and F_1 score results for RFCN using K-fold validation

The F_1 score results for all 5 groups are greater than 75%, which implies that more than $\frac{3}{4}$ of the detections were matched to the ground truths. The highest F_1 score was achieved by the models trained on Group B and C, both achieving 83%. There is a difference of 10 flagged FP detections between them, where Group B had less flags. This implies that Group B performed better in picking up the ground-truth firearm location than Group C. Group E had the smallest F_1 score of 77% due to its higher FP detection count, as it often misrepresented other similar looking objects for firearms. The overall F_1 score achieved by RFCN is 81%, while the average precision and recall scores are 73% and 91% respectively. Samples of the predictions made are displayed in Figure 6.5.

Loss values

RFCN recorded its classification, localisation, and total loss values during testing for the models trained on groups A, B, C, D, and E. Results have been detailed in Table 6.8.

Group	Classification loss	Localisation Loss	Total Loss
A	0.07	0.13	0.21
B	0.05	0.12	0.21
C	0.06	0.12	0.20
D	0.08	0.06	0.15
E	0.12	0.23	0.40
Average	**0.08**	**0.13**	**0.23**

Table 6.8: Loss values obtained for RFCN trained model

All loss values computed for RFCN are less than 0.5, which is a contributing factor towards attaining high accuracies. This shows that the summed errors for all loss values are small enough to visually achieve accurate predictions. The classification losses range between



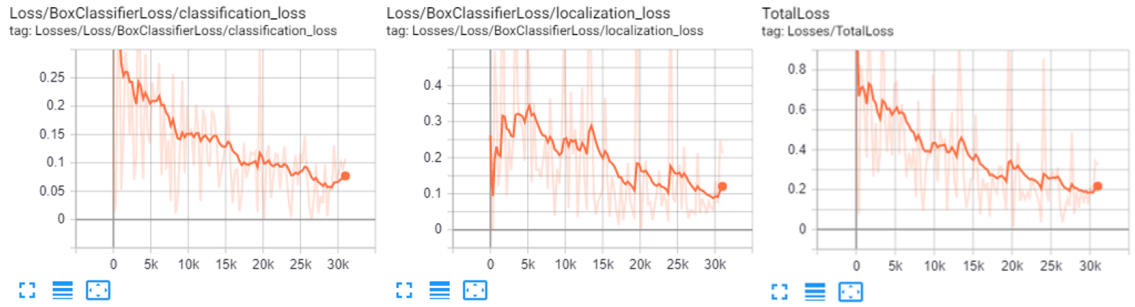
Figure 6.5: RFCN firearm detection samples taken from results obtained by Group B. In these images, all guns were detected accurately. The confidence score of the top-right and bottom-left images is lower than those scored in other images due to the small size of the firearms. This shows that RFCN is able to better process larger sized firearms than those at a small scale.

0.05 and 0.12, giving RFCN an average classification loss of 0.076. The localisation losses range from 0.06 and 0.23 to produce an average of 0.132. The classification losses are generally lower than the localisation losses. This effectively shows that the models perform better at picking up on the class type during testing, however do not perform as well in picking up its location. This can be seen and confirmed by the higher values of FP detections predicted by RFCN. The total loss ranges between 0.15 and 0.21, therefore producing an average total loss of 0.234 for RFCN, which is a low and stable result overall.

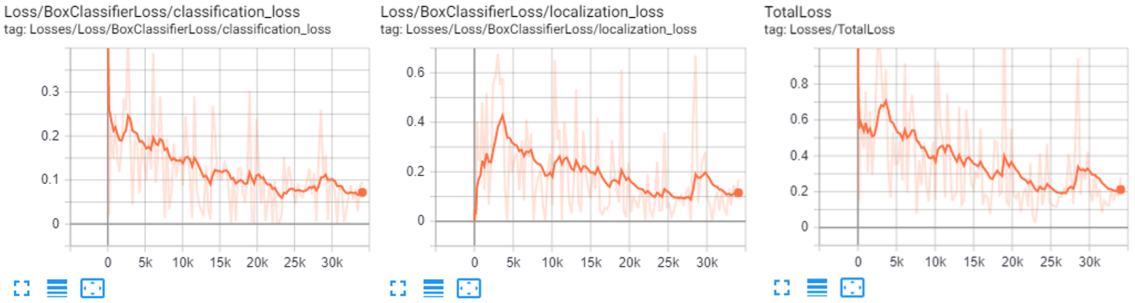
The lowest classification and localisation loss values have been attained by Groups B and D consecutively. Therefore, Group B performed best in detecting the correct class of objects in an image (also confirmed in Table 6.7 by the number of TP and TN detections), while Group D performed best in locating the firearm on the image, resulting in Group D achieving the highest number of TP detections with little error. The highest classification and localisation values were attained by Group E, which indicates that it had the highest rate of errors during detection.

The lowest total loss was produced by Group D, indicating that Group D overall had the least amount of errors during detection. On the other hand, Group E had the highest overall total loss therefore achieving the lowest performance. The training underwent by Group E was approximately an hour less than the training done on the other groups. To overcome this error, Group E needed to have underwent longer training periods in order to learn and process better from the training dataset. However, the loss value for RFCN overall is still a competitive 0.234 tending close to 0, which is an ideal loss value for a well trained model.

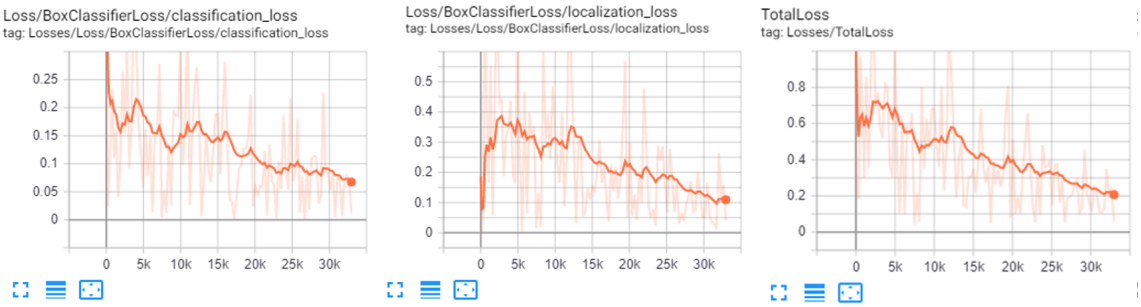
Figure 6.6 displays the increase in performance during training. Each graph had a downward slope shape indicating that the error reduced to a point at which it then began to stabilise and tend towards 0.



(a) RFCN Group A Loss Graphs (from left to right) Classification loss, Localisation loss, and Total loss

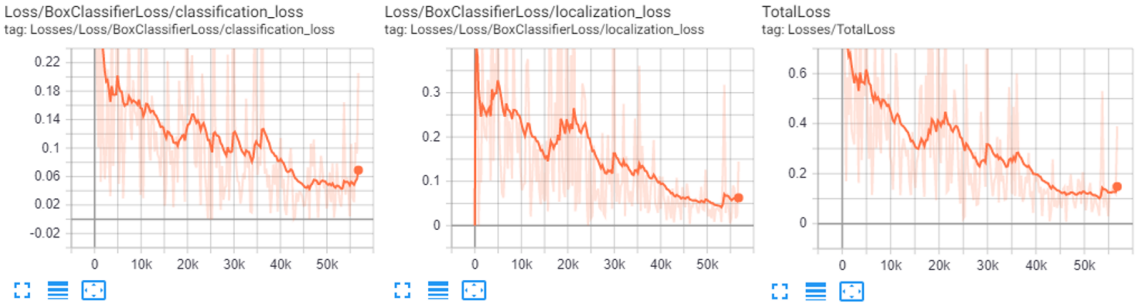


(b) RFCN Group B Loss Graphs (from left to right) Classification loss, Localisation loss, and Total loss



(c) RFCN Group C Loss Graphs (from left to right) Classification loss, Localisation loss, and Total loss

Figure 6.6: RFCN loss graphs



(d) RFCN Group D Loss Graphs (from left to right) Classification loss, Localisation loss, and Total loss

Figure 6.6: RFCN loss graphs

Inferencing Speed

The average speed of processing an image from the point at which the image is read into the trained classifier, to the actual display of its prediction upon the image is 1.85 seconds. The inferencing speed for each group is displayed in Table 6.9. The range of inferencing speeds lies between 1.7 and 2 seconds, which is quite fast for a RoI generation architecture. This version of RFCN makes use of the ResNet network which has the ability to skip over layers, therefore enhancing its speed. Secondly, the network has been designed to not have learnable layers following after the RoI layers, therefore significantly reducing computational costs which in turn positively impacts the speed of RFCN.

Group	Inferencing Speed (sec)
A	2.00
B	2.07
C	1.72
D	1.88
E	1.91
Average	**1.85**

Table 6.9: Inferencing speed of RFCN

RFCN Analysis

The average F_1 score achieved by RFCN is 81%. It occasionally misrepresented objects such as drills and hairdryers as guns due to the similar shape they have with firearms. This implies that RFCN relies heavily on the shape feature to determine whether or not an object fits into the gun class. However, the RFCN model performed well in identifying large and small sized guns, and detection was not negatively impacted by the picture resolution. The average recall amount at 91% is higher than the 73% score for precision. This indicates that the RFCN model detected a high proportion of TP ground truths within the gun class which increased the recall. In comparison, the FP count negatively impacted precision by lowering its value. This is as a result of misrepresenting similar looking gun objects as part of the gun class.

The average classification, localisation and total loss attained by RFCN is 0.076, 0.132, and 0.234 respectively. This shows that RFCN had a low error rate in detecting the gun and no-gun classes. The number of FP detections superseded the number of FN detections up to 3 times more, which reflected on the localisation error related to finding the location of the gun. However, both FP and FN detections are considerably lower than the TP and TN detections, which implies a low overall total loss. This visually translates into fewer occurrences of errors while making predictions.

The inference speed of the RFCN is determined by the architectural design of the model itself. RFCN is fully convolutional, therefore designed to distribute the computational processes throughout the network. As a result, the load is shared to reduce the processing times. Secondly, the learnable layers do not follow after the RoI layers, which reduces computation cost and saves processing time. Although RFCN makes use of RoI generation, these intentional additions to the RFCN design positively contributed towards the low average inference speed of 1.85 seconds.

6.2.7 Faster R-CNN Observations

Precision, recall, and F_1 score

According to the Tensorflow Detection Model Zoo GitHub page, Faster R-CNN resnet 101 model has an mAP of 32 [Rathod and Shi]. The model underwent K-fold cross validation to retrieve the results shown in Table 6.10.

Group	TP	FP	TN	FN	Precision	Recall	F_1 score
A	101	32	89	11	0.76	0.90	0.82
B	103	28	91	6	0.79	0.94	0.86
C	104	44	87	8	0.70	0.93	0.80
D	94	29	102	12	0.75	0.82	0.81
E	106	28	93	5	0.79	0.95	0.87
				Average:	**0.75**	**0.90**	**0.83**

Table 6.10: Precision, Recall, and F_1 score results for Faster R-CNN using K-fold validation

The average precision and recall is respectively 75% and 90% shared between Groups A, B, C, D, and E. Group B and E both share the highest precision at 79%, while Group C had the lowest score of 70%. This implies that both Groups B and E had the highest proportion of correct positive predictions out of all the predictions made, while Group C had a smaller proportion of accurate positive predictions. For recall, the highest percentage is 95% which was achieved by Group E, while the lowest at 82% was scored by Group D. This indicates that Group E was able to identify the most amount of ground truths known, as opposed to Group D which had the lowest performance. All groups managed to achieve an F_1 score greater than 80%, which portrays an overall high accuracy. Hence, the Faster R-CNN model obtained an average F_1 score of 83%. Samples of the detections achieved by Faster R-CNN are displayed in Figure 6.7.



Figure 6.7: Faster R-CNN firearm detection samples taken from the results of Group E. Due to the use of RoI pooling and anchor boxes, this classifier has the added advantage of being able to detect firearms at varying scales. As a result, it was able to accurately predict all firearms present in these samples with an accompanying high confidence score of 99%, which is higher than those predicted by RFCN for the smaller firearms.

Loss values

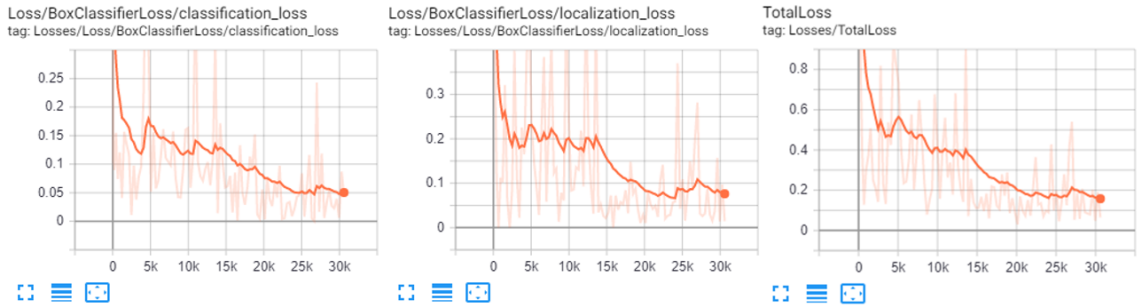
The classification, localisation and total loss values have been recorded for each group during testing. The results are displayed in Table 6.11. The classification losses are significantly low, with most of them hovering below 0.05. The highest classification loss of 0.16 was scored by Group D, while the lowest loss of 0.03 was scored by Group E. Based on these results, Group E has the smallest classification error in the whole group, which ties together with its detection performance where it achieved the highest number of TP detections, and second highest TN detections. Conversely, Group D had the lowest count of TP detections. The anomaly however is the TN detection count of Group D which happens to be the highest count attained. In such a case, it is possible that the classification error may have instead been influenced by the higher number of FN detections which has the highest count of all groups. The average classification loss for Faster R-CNN is 0.076.

Group	Classification loss	Localisation Loss	Total Loss
A	0.05	0.08	0.17
B	0.04	0.05	0.11
C	0.06	0.12	0.20
D	0.16	0.25	0.60
E	0.03	0.04	0.08
Average	**0.07**	**0.13**	**0.23**

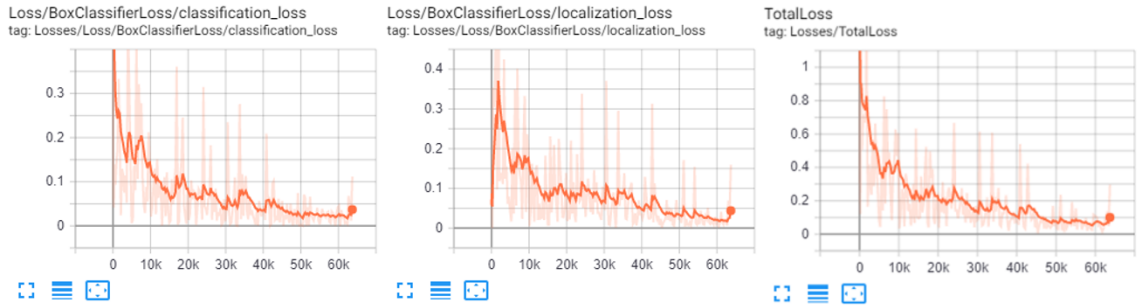
Table 6.11: Loss values obtained for Faster R-CNN trained models

The localisation losses are all below 0.5, with the highest value being 0.25 attained by Group D, and the lowest value by Group E. Group E also achieved the best performance in the localisation task by having the smallest error during detection, which confirms the low FN and FP detection count. The overall average localisation loss is valued at 0.132.

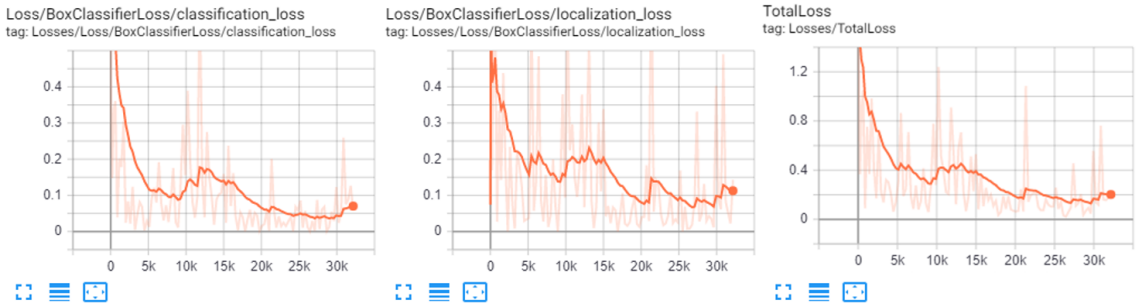
The total loss accounts for all types of errors encountered by the model. The highest total loss was attained by Group D at 0.6, while the lowest and best loss was attained by Group E valued at 0.08. This result indicates that Group E had the best performance generally in terms of classification and localisation, which can be confirmed by the F_1 score it achieved. The average total loss achieved by Faster R-CNN is 0.232. The loss values during training were recorded for all five groups. This has been detailed in Figure 6.8.



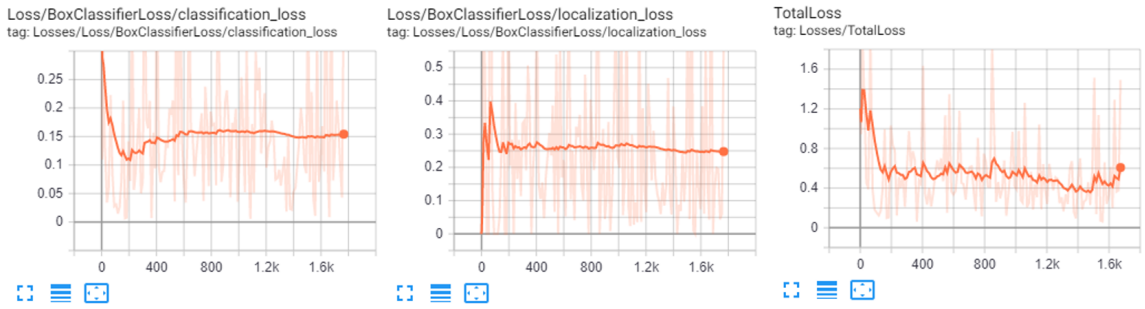
(a) Faster R-CNN Group A Loss Graphs (from left to right) Classification loss, Localisation loss, and Total loss



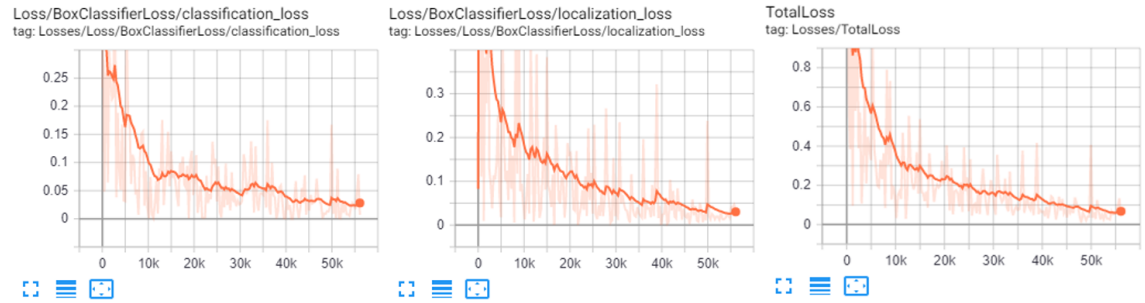
(b) Faster R-CNN Group B Loss Graphs (from left to right) Classification loss, Localisation loss, and Total loss



(c) Faster R-CNN Group C Loss Graphs (from left to right) Classification loss, Localisation loss, and Total loss



(d) Faster R-CNN Group D Loss Graphs (from left to right) Classification loss, Localisation loss, and Total loss



(e) Faster R-CNN Group E Loss Graphs (from left to right) Classification loss, Localisation loss, and Total loss

Figure 6.7: Loss graphs produced during training for Faster R-CNN

Inferencing Speed

Region Proposal classifiers tend to perform detections slower than the single stage object detectors due to the extra processing at each proposed generated region. Displayed in Table 6.12 are the inference speeds achieved by Faster R-CNN for all 5 groups. As expected from a heavily-weighted model, the average inference speed went up to 5.62 seconds. The fastest inference speed was achieved by both Groups A and B at 5.38 seconds, while the slowest model trained on Group E achieved an average speed of 6.23 seconds

Group	Inferencing Speed (sec)
A	5.38
B	5.38
C	5.51
D	5.62
E	6.23
Average	**5.62**

Table 6.12: Inferencing speed of Faster R-CNN

The dimensionality of images had a notable influence over the speed of inferencing. Images with dimensions greater than 1000×1000 took longer to process. A larger number of multiple proposed regions are generated at a time to scale the whole image, therefore

increasing the processing time before generating a prediction.

Faster R-CNN analysis

The average F_1 score achieved by Faster R-CNN is valued at 83%. This translates into Faster R-CNN having a high accuracy of detections. Misclassification therefore only occurs 17% of the time. The average recall value of 90% is higher than the precision value of 75%. This indicates that Faster R-CNN had a high proportion of predicted True Positives out of the pool of known ground truths. However, since the average recall is slightly lower, the proportion of accurate predicted positives out of all positive predictions was not as great. Despite this difference, the classifier handled the firearm detection task well. The FN detections were relatively low, indicating that the model is able to determine the difference between the gun class and background information. However, the FP detections were slightly higher, which implies that Faster R-CNN is more inclined to classify objects sharing similar firearm characteristics as guns. One noticeable issue during detection was the ability for the model to accurately detect more than 3 guns within the image where the guns are closely packed next to one another. This could indicate that the training set consisted mostly of lone guns per image, therefore giving the Faster R-CNN model a focus on just detecting 4 or less firearms within an image, and disregarding all others.

The average classification, localisation and total loss obtained by Faster R-CNN are 0.076, 0.132, and 0.234 consecutively. The low classification loss implies that the errors made during the classification task are relatively low, therefore giving this model a good prediction performance. The localisation loss is not far behind. The summed error is quite low, which implies that the model performed well at correctly locating the gun objects. The total loss generalized the classification and localization loss, giving the model a score of 0.234. This translates into a low error rate, therefore giving Faster R-CNN an overall good performance. This can also be confirmed by the low FN detections, and the generally low FP detections obtained by most of the groups.

Deep learning models utilising the region proposal generation method are known to have heavy computations which have a negative impact on the speeds. Faster R-CNN is one of those models, and as a result did not manage to achieve real-time speeds during inferencing. The average inferencing speed for Faster R-CNN is 5.62 seconds. The range of inferencing speeds obtained by the individual groups are between 5.38 and 6.23, which confirms the drastic difference between Faster R-CNN speeds and real-time speeds.

6.3 Comparison of results

6.3.1 Comparison of Single stage object detectors

The two single stage object detectors that have been trained for firearm detection are SSD inception V2 and YOLOv3. This section will thoroughly compare the results achieved by single stage object detectors in an effort to determine which one is better suited for the firearm detection task.

F_1 score comparison

The F_1 scores for SSD and YOLO are 77% and 75% respectively on the test set. This obviously indicates that SSD performed 2% better than YOLO in the firearm detection task. However, their visual results were very similar to one another. Both models struggled to distinguish between background and firearm objects when the colours match closely with one another. This stipulates that the single stage object detectors place heavy emphasis on the colour of the objects within each image. However, where the contrast between background and firearm object is large, the models were both able to easily classify the object as a gun and correctly find its location within the image as seen from Figures 6.1 and 6.3. Out of the 6 sampled images in Figures 6.1 and 6.3, similar FN predictions were made by SSD and YOLO, where the gun blended into the background due to its colour closely matching its surrounding. Images with better lighting or taken during the day seemed to have accurate classification and localisation predictions for both YOLO and SSD. Improvements to the model can be achieved by gathering more images taken during the night where firearms are present, or where the background and firearm colours are similar to one another. This will help the models to learn how to focus on other characteristics of the gun when faced with a situation where the colours in the image are closely matched. SSD had a tendency of overlooking small sized firearms in the image by associating them with the background. This negatively impacts the performance upon images in which the perspective of gun placement seems further away in the scene. Despite this, the SSD model picked out more detections from the pool of gun class ground truths than the YOLO model, and equally more from the background class ground truths based in its number of TN detections. This implies that SSD generally has a better firearm detection performance.

Loss comparison

SSD and YOLO achieved a total/overall loss of 1.78 and 0.4567 on the validation set. The higher total loss achieved by SSD is on account of the different types of losses that the classifier is factoring in. For the purposes of this research, only the classification and localisation are deemed relevant for the firearm detection task. The average classification and localisation loss achieved by SSD is 1.22 and 0.23 respectively. Both classifiers were able to achieve losses that sloped down to a steady decreasing level. However, better results could have been achieved by SSD if the model had been trained long enough for the loss to tend closer to 0. Gathering a collection of images taken at night, or where the background and firearm object colour matches closely could help to train the classifiers on how to differentiate between firearms and other objects of similar colour. However, based on the overall loss results, YOLO had the smallest summed error.

Inferencing speed comparison

The average inferencing speed of SSD and YOLO is 0.386 seconds and 0.401 seconds consecutively. Due to the architectural design of the single stage object detectors, both models were able to save a significant amount of computation time, therefore allowing them to perform detections at near real-time speeds. The SSD model achieved faster speeds than YOLO overall due to its structure, as heavy emphasis of its design was placed on reducing computational time to allow for inferencing to match real-time speeds. Smaller dimensions of images would definitely increase the speed of inferencing, however smaller pixel sizes

run the risk of losing useful visual information, which could have a negative impact on detection. Therefore, a trade-off would have to be established here in which the image sizes cannot be made any smaller than 94×206, which is the smallest image dimension in the dataset.

6.3.2 Comparison of Two stage object detectors

In this section, the two stage object detectors are being compared with one another for the firearm detection task. The average performance of both models are compared with one another, and a discussion around possible means of improving the performance is also included.

F_1 score comparison

Faster R-CNN achieved the highest F_1 score of 83% on the test set compared to RFCN scoring 81%. The detection performance of Faster R-CNN indicates that the architectural design of the model is more heavily focused on achieving accurate results in comparison to RFCN. Both models apply RoI pooling on feature maps to extract relevant information for detection, which helps both models to achieve higher detection accuracies. However, Faster R-CNN uses anchor boxes to handle detecting firearms at various scales, therefore giving it the edge over RFCN in being able to pick up on even the tiniest of firearms within an image. Based on the visual results shown in Figures 6.5 and 6.7, both Faster R-CNN and RFCN were able to correctly detect firearms in all sample images. However, Faster R-CNN produced higher confidence score values overall when making a prediction. This proves that added steps in the detection process such as the use of anchor boxes helps the classifier more accurately decipher between a firearm and non-firearm object.

Loss comparison

The average total loss of Faster R-CNN is 0.232 on the validation set, while RFCN attained a value of 0.234. There is a very slight difference between the two models summed errors. This can be associated with the similarities between the model architectures in which multiple regions of interest are generated where the firearm object is most likely to be located. This enhanced the performance of both models, which in turn decreased the likelihood of error detections in terms of classification and localisation simultaneously. Both models managed to achieve a limited amount of FN and FP detections due to their lower loss values.

Inferencing speed comparison

RFCN performed inferencing at an average speed of 1.85 seconds, while Faster R-CNN achieved this in 5.62 seconds. RFCN managed to achieve much higher inferencing speeds than Faster R-CNN because of the placement of learnable layers in the network. These layers are not positioned next to the RoI layers, therefore allowing for less processing which translates into faster output generation. Secondly, the RFCN model is fully convolutional which allows for the work required to generate a prediction to be distributed. In summary, the architectural design of RFCN gives it the advantage over Faster R-CNN because of the design tweaks which have been included to eliminate unnecessary processing times.

6.3.3 Classifier Results

Classifier performances have been represented in Table 6.13 which includes the overall F_1 score on the test set, total loss on the validation set, and the average inference speed for each classifier. The average of five groups per classifier have been used to represent the true performance of each classifier in order to attain realistic results from training on a small dataset.

Model Name	**F_1 score (%)**	**Inferencing speed (sec)**	**Loss**
SSD inception v2	77	0.03	1.78
RFCN ResNet101	81	1.85	0.23
YOLOv3	75	0.04	0.45
Faster R-CNN ResNet101	83	5.62	0.23

Table 6.13: Model performance output

6.4 Answers to research questions

1. **What are the state-of-the-art deep learning object detection methods that can be used for vision-based firearm detection?**
   A series of deep learning tools have been developed to assist with the object detection task, such as those with the ability to learn semantic high-level deep features added onto traditional models [Zhao *et al.* 2019]. Various object detection architectures have been modified in an effort to improve on their performance of object localisation and classification.

   Extracting informative features to recognize different objects is one of the necessary tasks needed to obtain robust representation of what a firearm looks like. This helps with the training process to have the model learn visual cues that indicate when a firearm has been sited. However, conditions to do with various types of background noise, illumination, and object orientation make the task more difficult to carry out with robustness. Generic object detection is a category of detection methods that are designed to classify and locate existing objects in an image, and therefore are suited to the firearm detection task. Models which undergo Region-based training with CNN features have deep architectures which can handle learning complex features for the firearm detection task. R-CNN made way for the development of models such as Fast and Faster R-CNN which have an almost cost-free RPN infused in them for sharing convolutional features based on the whole input image with the network. Some of the advantages of CNN-based architectures that have made them state-of-the-art include:

   - The ability to tackle high-dimensional computer vision tasks and solving them from different orientation due to its complex deep learning strength.
   - Being able to mutually optimize multiple tasks related to object detection such as the combining of classification and bounding box regression processes into a multi-task process.

- The ability to handle features of multiple representations in hierarchical format all the way from pixel level to features of higher semantic value using an automatic multi-stage process of investigation.

One stage object detectors incorporate the classification and bounding box regression tasks at decreased computational costs via a mapping from pixel level to class probabilities and predicted bounding box coordinates. Architectures such as SSD and YOLO make use of this approach. SSD is constructed to use multiple feature maps handling varying resolutions which can deal with images containing objects of different sizes. YOLO achieves the same task using a fixed-grid regression approach, and has high FPS speeds which allow for near real-time detections. YOLOv3 has obtained higher accuracy results than its predecessors and has fast inferencing speeds compared to other object detection models, therefore giving it its state-of-the-art status.

2. **Which deep learning method performs fastest for detection of firearms on images?**
There are 4 different models which trained and tested on the firearm dataset used for this research. Their inferencing speeds have been recorded by computing average speed for all images per model. This produces the following results (from top to bottom in order of fastest to slowest):

Model Name	**Inference speed (sec)**
SSD inception v2	0.03
YOLOv3	0.04
RFCN ResNet101	1.85
Faster R-CNN ResNet101	5.62

Table 6.14: Comparison of model inferencing speeds

Based on the results gathered, YOLO is approximately 0.01 seconds slower than SSD during inferencing, therefore making SSD the faster model for releasing firearm detection prediction outputs. Both YOLO and SSD are significantly faster than the other tested models possibly due to the "light-weight" nature of their network structures. The localisation and classification tasks are performed in one stage rather than splitting tasks up into separate entities to be processed one after the other as is done with two-stage detectors which rely on the generation of feature maps in the first stage before RoI proposals can be produced in the second stage for subsequent analysis. Classifiers like Faster R-CNN consisting of deep base networks like ResNet101 require more training time than a classifier such as SSD inception v2 as its base network has fewer layers.

3. **What is the best performing deep learning method in terms of high accuracy for firearm detection based on its overall F_1 score?**
The F_1 score for each model takes into consideration both precision and recall values recorded to determine the accuracy of detection. Models which produced better recall values include RFCN ResNet101, Faster R-CNN ResNet 101, SSD inception

v2, while YOLOv3 produced better precision values. The difference between precision and recall per model had a direct impact on the F_1 scores. Based on the results obtained, it is apparent that the models which obtained higher recall values managed to also produce higher F_1 score values than their counterpart models with higher precision values. This directly implies that they have higher accuracy results. Detailed below are the F_1 scores per model:

Model Name	F_1 **scores (%)**
Faster R-CNN ResNet101	83
RFCN ResNet101	81
SSD inception v2	77
YOLOv3	75

Table 6.15: Comparison of model F_1 scores

Based on the details presented in Table 6.15, the region proposal based classifiers all outperformed the regression based models in terms of accuracy. Out of these better performing models, the model which is more accurately able to detect firearms in an image is Faster R-CNN (resnet 101). This may partially be as a result of the hierarchical deep base network structure of Resnet101, where the first stage of the network called "conv_1" consists of 7×7 convolutional layer followed by a 3×3 pooling layer [Wu *et al.* 2019]. Thereafter, the building blocks contain 3 convolutional layers each with varying dimensionality. This approach may be better suited (compared to other tested models) for analysing firearm textures and shapes at different scales.

6.5 Conclusion

This chapter analyses the classifier performance by comparing and contrasting their speed and accuracy with one another. Each classifier is ranked based on how well it performed for speed of inferencing, as well as their individual F_1 scores translated into an accuracy value in percentage format, and the loss values attained during training. In the concluding chapter that follows, a summary of the major points around this dissertation are discussed.

Chapter 7

Conclusion

Crime is a growing source of concern in the nation of South Africa. Unfortunately, it has degenerated to a level of extreme violence involving the use of weapons. In many cases, criminals have been able to get away with physically injuring victims. Statistics in South Africa show that firearms are the first choice of weapons used for aggravated robberies. Due to this growing trend, it is important to put into place measures that will assist the SAPS and other security agencies in combating crime and keeping surrounding areas safe. The firearm recognition model has been proposed as part of a solution to demonstrate how computer vision and deep learning can be used as a security enhancement.

A supervised learning approach has been applied on the collected dataset for training pre-existing state-of-the-art deep learning models, which involves annotating images to indicate where the firearm is located. The selected learning algorithms to be compared with one another are SSD, YOLO, RFCN, and Faster R-CNN. The research conducted show that region proposal methods outperform the regression methods in terms of their accuracy in classifying and locating a firearm correctly in an image. However, in terms of speed of inferencing, the regression models still prevailed as being the faster models that are able to detect in near real-time speeds. Differences in behaviour are directly influenced by backbone network structures and delegation of tasks required for detection. Deeper networks have longer training time due to the number of convolutional layers that the data passes through for processing of features. The complexity of two-stage architectures allows for firearms of varying sizes to be processed due to the multi-scale manner in which they are able to extract features. The single stage detectors in contrast are designed for inferencing at fast speeds as a result of utilising light-weight network structures for a single forward pass to handle classification and localisation.

Two-stage object detectors like Faster R-CNN may be better suited to handle the task of accurately detecting firearms, however the single-stage object detectors are more consistent with real-time speeds for scenarios where surveillance cameras are deployed for security purposes requiring immediate action. To further improve on firearm detection, it is suggested that deep learning methods such as Faster R-CNN are modified in such a way that the computational heaviness is reduced while still maintaining its complexity towards analysing an image. Improvements can also be implemented on SSD to be able to detect firearms at multiple scales, and upon similar coloured backgrounds.

Various factors contributed in limiting the full performance capacity of each classifier.

Firstly, due to load shedding, training was intermittent causing the training period to be cut short. Secondly, due to limited access to labelled firearm detection datasets, the size of the dataset is relatively small. This could possibly lead to biased results that have not properly handled overfitting. Therefore, future work can involve significantly increasing the size of the dataset to be trained on for firearm detection. Firearm detection can also be extended to a real-time environment in which actual surveillance footage is put through the trained classifiers to test their performance.

References

[Abadi *et al.* 2015] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, 2015. Software available from tensorflow.org.

[Albani *et al.* 2017] Dario Albani, Ali Youssef, Vincenzo Suriani, Daniele Nardi, and Domenico Bloisi. A deep learning approach for object recognition with nao soccer robots. pages 392–403, 11 2017.

[Alqubaa and Tian 2012] Abdalrahman Alqubaa and Gui Yun Tian. Weapon detection and classification based on time-frequency analysis of electromagnetic transient images. *International Journal of Advances in Systems and Measurements*, 3(3 & 4), 2012.

[Attorneys 2017] Christo Mulder Attorneys. *Legally owning a firearm in South Africa*, October 2017. Retrieved 20 May, 2019, from `https://www.muldesrattorneyss.co.za`

[Bay *et al.* 2006] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[Bisong 2019] Ekaba Bisong. *Google Colaboratory*, pages 59–64. 09 2019.

[Bodla *et al.* 2017] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S Davis. Soft-nms–improving object detection with one line of code. In *Proceedings of the IEEE international conference on computer vision*, pages 5561–5569, 2017.

[Boylen 2019] Boylen. *How metal detectors work*, 2019. Retrieved 25 September, 2020, from `https://www.minelab.com/knowledge-base/getting-started/how-metal-detectors-work`

[Brownlee 2019] Jason Brownlee. *How to Visualize Filters and Feature Maps in Convolutional Neural Networks*, May 2019. `https://machinelearningmastery.com/how-to-visualize-filters-and-feature-maps-in-convolutional-neural-networks/`.

[Brownlee 2020] Jason Brownlee. *What is deep learning*, 2020.

[Calonder *et al.* 2010] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

[Castillo *et al.* 2019] Alberto Castillo, Siham Tabik, Francisco Pérez, Roberto Olmos, and Francisco Herrera. Brightness guided preprocessing for automatic cold steel weapon detection in surveillance videos with deep learning. *Neurocomputing*, 330:151 – 161, 2019.

[Cen 2016] Kaiqi Cen. *Study of Viola-Jones Real Time Face Detector*, 2016.

[Check 2020] Africa Check. *FACTSHEET: South Africa's crime statistics for 2019/20*, August 2020. `https://africacheck.org/factsheets/factsheet-south-africas-crime-statistics-for-2019-20/`.

[Choi *et al.* 2016] Keunwoo Choi, George Fazekas, and Mark Sandler. Explaining deep convolutional neural networks on music classification. *arXiv preprint arXiv:1607.02444*, 2016.

[Christiansen 2018] Anders Christiansen. *Anchor Boxes — The key to quality object detection*, October 2018. Retrieved 26 June, 2020, from `https://medium.com/@andersasac/anchor-boxes-the-key-to-quality-object-detection`

[Coetzee 2004] Ben Coetzee. Identifying the culprits: tracking the circulation of small arms. *Focus on Arms in Africa*, 3(1):3–5, 2004.

[Dai *et al.* 2016] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.

[Deng *et al.* 2009] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[Egiazarov *et al.* 2019] Alexander Egiazarov, Vasileios Mavroeidis, Fabio Massimo Zennaro, and Kamer Vishi. Firearm detection and segmentation using an ensemble of semantic neural networks. In *2019 European Intelligence and Security Informatics Conference (EISIC)*, pages 70–77. IEEE, 2019.

[eNCA 2018] eNCA. *REVEALED: Weapons of choice for murder in SA*, 2018.

[Everingham and Winn 2011] Mark Everingham and John Winn. The pascal visual object classes challenge 2012 (voc2012) development kit. *Pattern Analysis, Statistical Modelling and Computational Learning, Tech. Rep*, 8, 2011.

[Everingham *et al.* 2007] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge 2007 (voc2007) results. 2007.

[Fu *et al.* 2017] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C. Berg. DSSD : Deconvolutional single shot detector. *CoRR*, abs/1701.06659, 2017.

[Gandhi 2018] Rohith Gandhi. *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms*, 2018.

[Geethapriya *et al.* 2019] S. Geethapriya, N. Duraimurugan, and S.P. Chokkalingam. Real time object detection with yolo. *International Journal of Engineering and Advanced Technology (IJEAT)*, 8, February 2019.

[Girshick *et al.* 2014] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[Girshick 2015] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[Grel 2017] Tomasz Grel. *Region of interest pooling explained*, February 2017. Retrieved 26 June, 2020, from `https://deepsense.ai/region-of-interest-pooling-explained/`

[Grüne 2020] Ansgar Grüne. *What's a good F1 score?*, 2020.

[He *et al.* 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[Hechun and Xiaohong 2019] Wang Hechun and Zheng Xiaohong. Survey of deep learning based object detection. In *Proceedings of the 2Nd International Conference on Big Data Technologies*, ICBDT2019, pages 149–153, New York, NY, USA, 2019. ACM.

[Hollemans 2018] Matthijs Hollemans. *One-stage object detection*, June 2018. Retrieved 9 April, 2020, from `https://machinethink.net/blog/object-detection/`

[Hosang *et al.* 2017] Jan Hendrik Hosang, Rodrigo Benenson, and Bernt Schiele. Learning non-maximum suppression. *CoRR*, abs/1705.02950, 2017.

[Howard *et al.* 2017] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[Hui 2018] Jonathan Hui. *Understanding Region-based Fully Convolutional Networks (R-FCN) for object detection*, March 2018. `https://medium.com/@jonathan_hui/understanding-region-based-fully-convolutional-networks-r-fcn-for-object-detection`.

[IMFDB 2007] IMFDB. *Internet Movie Firearms Database*, 2007.

[Jiao *et al.* 2019] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. A survey of deep learning-based object detection. *IEEE Access*, 7:128837–128868, 2019.

[Jing *et al.* 2020] Junfeng Jing, Dong Zhuo, Huanhuan Zhang, Yong Liang, and Min Zheng. Fabric defect detection using the improved yolov3 model. *Journal of Engineered Fibers and Fabrics*, 15:1558925020908268, 2020.

[Kamal 2019] Amro Kamal. *Study of Viola-Jones Real Time Face Detector*, 2019.

[Kanehisa and Neto 2019] R Kanehisa and A Neto. Firearm detection using convolutional neural networks. In *Proceedings of the 11th International Conference on Agents and Artificial Intelligence, ICAART*, volume 2, pages 707–714, 2019.

[Kauderer-Abrams 2017] Eric Kauderer-Abrams. Quantifying translation-invariance in convolutional neural networks. *arXiv preprint arXiv:1801.01450*, 2017.

[Keegan 2005] Margaret Keegan. *The proliferation of firearms in South Africa, 1994-2004*. Gun Free South Africa Braamfontein, South Africa, 2005.

[Khazri 2019] Achraf Khazri. *Faster RCNN Object detection*, April 2019. Retrieved 26 June, 2020, from `https://towardsdatascience.com/faster-rcnn-object-detection`

[Krizhevsky *et al.* 2017] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[Lai and Maples 2017] Justin Lai and Sydney Maples. *Developing a Real-Time Gun Detection Classifier*, 2017.

[Larxel 2020] Larxel. *Handgun Detection - Improve public safety by detecting handguns within images*, 2020.

[Le Huy Hien and V.H. 2020] Ngo Le Huy Hien and Nguyen V.H. Recognition of plant species using deep convolutional feature extraction. 11:904–910, 06 2020.

[Limonova *et al.* 2020a] Elena Limonova, Daniil Alfonso, Dmitry Nikolaev, and Vladimir V Arlazarov. Resnet-like architecture with low hardware requirements. *arXiv preprint arXiv:2009.07190*, 2020.

[Limonova *et al.* 2020b] Elena Limonova, Daniil Matveev, Dmitry Nikolaev, and Vladimir V Arlazarov. Bipolar morphological neural networks: convolution without multiplication. In *Twelfth International Conference on Machine Vision (ICMV 2019)*, volume 11433, page 114333J. International Society for Optics and Photonics, 2020.

[Lin *et al.* 2013] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[Lin *et al.* 2014] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[Lin *et al.* 2017] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[Linh and Arai 2019] Tran Duy Linh and Masayuki Arai. Two-stage deep neural network for general object detection. *Journal of Information Processing*, 27:268–277, 2019.

[Liu *et al.* 2015] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[Liu *et al.* 2016] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander Berg. Ssd: Single shot multibox detector. volume 9905, pages 21–37, 10 2016.

[Liu *et al.* 2018] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. *Deep Learning for Generic Object Detection: A Survey*, 2018.

[Lokanath *et al.* 2017] M Lokanath, K Kumar, and E Keerthi. Accurate object classification and detection by faster-rcnn. *IOP Conference Series: Materials Science and Engineering*, 263:052028, 11 2017.

[Mecanico] Lojado Mecanico. *Professional Impact Drill with Variable Speed 550W - BOSCH-GSB550RE.* `https://www.lojadomecanico.com.br/produto/92104/21/221/furadeira-de-impacto-profissional-com-velocidade-variavel-550w-110v-bosch-gsb550re`.

[Nagtegaal 2018] Advocate Nagtegaal. *6 questions answered about gun ownership*, October 2018. Retrieved 22 May, 2019, from `https://www.news24.com/Analysis`

[Nash *et al.* 2018] Will Nash, Tom Drummond, and Nick Birbilis. A review of deep learning in the study of materials degradation. *npj Materials Degradation*, 2(1):1–12, 2018.

[Noh *et al.* 2018] Junhyug Noh, Soochan Lee, Beomsu Kim, and Gunhee Kim. Improving Occlusion and Hard Negative Handling for Single-Stage Pedestrian Detectors. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[Olmos *et al.* 2018] Roberto Olmos, Siham Tabik, and Francisco Herrera. Automatic handgun detection alarm in videos using deep learning. *Neurocomputing*, 275:66–72, 2018.

[Olmos *et al.* 2019] Roberto Olmos, Siham Tabik, Alberto Lamas, Francisco Pérez-Hernández, and Francisco Herrera. A binocular image fusion approach for minimizing false positives in handgun detection with deep learning. *Information Fusion*, 49:271–280, 2019.

[O'Mahony *et al.* 2019] Niall O'Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. Deep learning vs. traditional computer vision. In *Science and Information Conference*, pages 128–144. Springer, 2019.

[Pedregosa *et al.* 2011] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Poirson *et al.* 2016] Patrick Poirson, Phil Ammirato, Cheng-Yang Fu, Wei Liu, Jana Kosecka, and Alexander C Berg. Fast single shot detection and pose estimation. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 676–684. IEEE, 2016.

[Rathod and Shi] Vivek Rathod and Yining Shi. *TensorFlow 1 Detection Model Zoo.*

[Redmon and Farhadi 2017] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[Redmon and Farhadi 2018] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[Redmon *et al.* 2016] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[Ren *et al.* 2015a] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[Ren *et al.* 2015b] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*, 2015.

[Saxena 2020] Pawan Saxena. *R-CNN vs Fast R-CNN vs Faster R-CNN — ML*, March 2020. Retrieved 25 June, 2020, from `https://www.geeksforgeeks.org/r-cnn-vs-fast-r-cnn-vs-faster-r-cnn-ml/`

[Saxena 2021] Shipra Saxena. *Introduction to Batch Normalization*, 2021.

[Shaikh 2018] Faizan Shaikh. *Deep Learning in the Trenches: Understanding Inception Network from Scratch*, 2018.

[Shi *et al.* 2019] Wenxu Shi, Shengli Bao, and Dailun Tan. Ffessd: An accurate and efficient single-shot detector for target detection. *Applied Sciences*, 9:4276, 10 2019.

[Shrivastava *et al.* 2016] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 761–769, 2016.

[Sicetsha 2018] Andile Sicetsha. *Crime stats South Africa 2018: the murder rate increased by 6.9%*, September 2018. Retrieved 20 May, 2019, from `http://www.thesouthafrican.com`

[Soviany and Ionescu 2018] Petru Soviany and Radu Tudor Ionescu. Optimizing the trade-off between single-stage and two-stage deep object detectors using image difficulty prediction. In *2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 209–214. IEEE, 2018.

[Soviany and Ionescu 2019] Petru Soviany and Radu Tudor Ionescu. Frustratingly easy trade-off optimization between single-stage and two-stage deep object detectors. In Laura Leal-Taixé and Stefan Roth, editors, *Computer Vision – ECCV 2018 Workshops*, pages 366–378, Cham, 2019. Springer International Publishing.

[Szegedy *et al.* 2015] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[Szegedy *et al.* 2016] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[Szegedy *et al.* 2017] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[Thakur 2019] Rohit Thakur. *Step by step VGG16 implementation in Keras for beginners*, 2019.

[Tiwari and Verma 2015] Rohit Kumar Tiwari and Gyanendra K. Verma. A computer vision based framework for visual gun detection using harris interest point detector. *Procedia Computer Science*, 54:703 – 712, 2015. Eleventh International Conference on Communication Networks, ICCN 2015, August 21-23, 2015, Bangalore, India Eleventh International Conference on Data Mining and Warehousing, ICDMW 2015, August 21-23, 2015, Bangalore, India Eleventh International Conference on Image and Signal Processing, ICISP 2015, August 21-23, 2015, Bangalore, India.

[Triguero *et al.* 2017] Francisco Triguero, Siham Tabik, Alberto Castillo Lamas, Francisco Pérez Hernández, and Roberto Olmos Pimentel. *Weapons detection for security and video surveillance*, 2017.

[Tyagi 2019] Deepanshu Tyagi. *Introduction to SIFT(Scale Invariant Feature Transform)*, 2019.

[Tzutalin 2015] Tzutalin. *LabelImg*. Free Software: MIT License, 2015.

[Vabalas *et al.* 2019] Andrius Vabalas, Emma Gowen, Ellen Poliakoff, and Alexander J Casson. Machine learning algorithm validation with a limited sample size. *PloS one*, 14(11):e0224365, 2019.

[Velocidad] Metal Velocidad. *Uzi*. Retrieved 7 August, 2020, from `https://sco.wikipedia.org/wiki/Uzi`

[Weng 2018] Lilian Weng. *Object Detection Part 4: Fast Detection Models*, December 2018. Retrieved 9 April, 2020, from `https://lilianweng.github.io/lil-log/2018/12/27/object-detection-part-4.html`

[Writer 2016] Businesstech Staff Writer. *These are the weapons SA criminals use to terrorize us*, 2016.

[Wu *et al.* 2019] Huiyan Wu, Ming Xin, Wen Fang, Hai-Miao Hu, and Zihao Hu. Multi-level feature network with multi-loss for person re-identification. *IEEE Access*, 7:91052–91062, 2019.

[Xie *et al.* 2017] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[Zhang and Zhang 2009a] Ethan Zhang and Yi Zhang. *Average Precision*, pages 192–193. Springer US, Boston, MA, 2009.

[Zhang and Zhang 2009b] Ethan Zhang and Yi Zhang. *Average Precision*, pages 192–193. Springer US, Boston, MA, 2009.

[Zhang *et al.* 2015] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):1943–1955, 2015.

[Zhao *et al.* 2019] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 30(11):3212–3232, 2019.