

https://dspace.org/

Comparing and contrasting the performance of single and two stage object detectors

Mphahlele, Kgaugelo

2022
https://hdl.handle.net/10539/33918

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

Comparing and contrasting the performance of single and two stage object detectors.

Kgaugelo MPHAHLELE
Student Number: 818 683

Supervisor: Dr. D. REDDY



A research report submitted in partial fulfillment of the requirements for the degree of Master of Science in the field of e-Science

in the

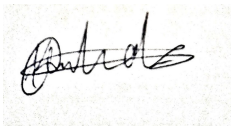
School of Computer Science and Applied Mathematics

University of the Witwatersrand, Johannesburg

8 February 2022

Declaration

I, Kgaugelo MPHAHLELE , declare that this research report is my own, unaided work. It is being submitted for the degree of Master of Science in the field of e-Science at the University of the Witwatersrand, Johannesburg. It has not been submitted for any degree or examination at any other university.



Kgaugelo MPHAHLELE

Student Number: 818 683

8 February 2022

Abstract

Object detectors are used in a wide array of different contexts and as such different implementations will vary in terms of performance and results. This study investigated how four object detectors perform on several datasets. More specifically and what is of particular interest would be the performance of these classifiers on remotely sensed images. The four object detectors that are ultimately implemented would be the Regions with Convolutional Neural Networks (RCNN), the Faster Regions with Convolutional Neural Networks (Faster RCNN), You Only Look Once (YOLO) and the Single Shot Multi-box Detector (SSD). In regards to remotely sensed images, it is ultimately found that the RCNN had the detection highest accuracy followed by Faster RCNN, YOLO and the SSD.

Acknowledgements

The steadfast support of my family and friends has helped provide me with the necessary and conducive environment to conduct this investigation. Furthermore I would like to thank Dr. D. Reddy for his steady guidance and supervision.

Contents

Declaration i

Abstract ii

Acknowledgements iii

List of Figures vii

List of Tables ix

List of Abbreviations xi

1 Introduction 1
- 1.1 Background . . . 1
- 1.2 Research Question (or Problem Statement) . . . 4
  - 1.2.1 Research Area . . . 4
  - 1.2.2 Research Problem . . . 4
  - 1.2.3 Research Aims and Objectives . . . 5
- 1.3 Limitations of this research investigation . . . 6

2 Literature Review 7
- 2.1 Regions with Convolutional Neural Network(RCNN) . . . 7
  - 2.1.1 The Architecture of the Model . . . 8
  - 2.1.2 Test Time Deduction for RCNN . . . 9
  - 2.1.3 Run-time Analysis for RCNN . . . 10
  - Supervised pre-Training . . . 11
  - Domain Fine Tuning of the RCNN . . . 11
  - Object Category Classifiers of the RCNN. . . . 11

2.2 Faster RCNN . . . 12
2.2.1 The Region Proposal Network . . . 13
Generating Anchors . . . 14
Translation-Invariant . . . 15
Multi-Scale Anchors . . . 15
Loss Function for Training RPNs . . . 16
Training Hhase for the RPN . . . 18
2.2.2 Exchanging Features between the Fast RCNN as well as the RCNN . . . 19
4-Step Alternating Training. . . . 20
Implementation Details . . . 21
2.3 YOLO . . . 22
2.3.1 Training the model with inputs of multiple scales . . . 28
2.4 SSD . . . 31
2.4.1 Model for the SSD . . . 32
2.4.2 Training for the SSD . . . 33

3 Research Methodology **37**
3.1 Introduction . . . 37
3.1.1 Research Design . . . 37
3.2 Methodology . . . 38
3.2.1 Proposed Method . . . 38
Methodology for RCNN . . . 38
Methodology for Faster RCNN . . . 43
Methodology for YOLO . . . 46
Methodology for SSD . . . 48
3.2.2 Datasets . . . 50
3.2.3 Analysis . . . 51
3.3 Technical Specifications . . . 54
3.4 Limitations . . . 54
3.5 Ethical Considerations . . . 55
3.6 Conclusion . . . 56

4 Results, Discussion and Limitations **57**
4.1 Results . . . 57

4.1.1 Results for the PASCAl VOC training dataset 57
4.1.2 Results for the Udacity self driving car training dataset 60
4.1.3 Results for the Eastern North American 24 dataset 62
4.1.4 Results for the Snapshot Serengeti training dataset 65
4.1.5 Results for the DOTA dataset 68
4.1.6 Consolidation of results 71
4.2 Discussion 72
4.2.1 Regions with Convolutional Neural Networks 73
4.2.2 Faster Regions with Convolutional Neural Networks 76
4.2.3 You Only Look Once 78
4.2.4 Single Shot MultiBox Detector 80

5 Conclusions and Future Work **82**
5.1 Conclusions 82
5.2 Future Work 83

Bibliography **84**

List of Figures

2.1 This figure illustrates the functionality of the RCNN [5]. 12
2.2 The figure illustrates a basic representation of the Faster RCNN implementation [11]. 14
2.3 The figure illustrates the implementation of the YOLO methodology at a basic level [12]. 24

3.1 This figure illustrates the model summary of the VGGNet that was used [29], [31]. 42

4.1 Bar Graph illustrating the mAP of each of the object detectors for the PASCAL VOC dataset [17]. 60
4.2 Bar Graph illustrating the mAP of each of the object detectors for the Udacity self driving car testing dataset [18]. 62
4.3 Bar Graph illustrating the mAP of each of the object detectors for the Eastern North American testing dataset [19]. 65
4.4 Bar Graph illustrating the mAP of each of the object detectors for the Serengeti Snapshot testing dataset [20]. 68
4.5 Bar Graph illustrating the mAP of each of the object detectors for the DOTA testing dataset [21]. 71
4.6 Bar graph denoting the object detectors across all the datasets. 1 denotes the Pascal VOC [17], 2 denotes the Udacity self driving car testing dataset [18], 3 denotes the Eastern Northern American 24 testing dataset [19], 4 denotes the Serengeti Snapshot testing dataset [20], and 5 denotes the DOTA testing dataset [21]. 72
4.7 An example of RCNN generating candidate regions on an image from PASCAL VOC [17]. 73

4.8 An example of RCNN generating candidate regions on an image from the Udacity self driving car dataset [18]. 74
4.9 An example of RCNN generating candidate regions on an image from the Eastern North American dataset [19]. 74
4.10 An example of RCNN generating candidate regions on an image from the snapshot Serengeti dataset [20]. 75
4.11 An example of RCNN generating candidate regions on an image from the DOTA dataset [21]. 75

List of Tables

4.1 Table with the average precision(%) results for each object class and the subsequent mean average precision(%) score for each object detector, with regard to the PASCAL VOC training dataset [17]. 58
4.2 Table with the average precision(%) results for each object class and the subsequent mean average precision(%) score for each object detector, with regard to the PASCAL VOC testing dataset [17]. 59
4.3 Table with the average precision(%) results for each object class and the subsequent mean average precision(%) score for each object detector with regard to the Udacity self driving car training dataset [18]. 61
4.4 Table with the average precision(%) results for each object class and the subsequent mean average precision(%) score for each object detector with regard to the Udacity self driving car testing dataset [18]. 61
4.5 Table with the average precision(%) results for each object class and the subsequent mean average precision(%) score for each object detector with regard to the Eastern North American training dataset [19]. 63
4.6 Table with the average precision(%) results for each object class and the subsequent mean average precision(%) score for each object detector with regard to the Eastern North American testing dataset [19]. 64
4.7 Table with the average precision(%) results for each object class and the subsequent mean average precision(%) score for each object detector with regard to the snapshot Serengeti training dataset [20]. . . 66
4.8 Table with the average precision(%) results for each object class and the subsequent mean average precision(%) score for each object detector with regard to the snapshot Serengeti testing dataset [20]. . . . 67

4.9 Table with the average precision(%) results for each object class and the subsequent mean average precision(%) score for each object detector with regard to the DOTA training dataset[21]. 69
4.10 Table with the average precision(%) results for each object class and the subsequent mean average precision(%) score for each object detector with regard to the DOTA testing dataset[21]. 70

List of Abbreviations

RCNN	**R**egions with **C**onvolutional **N**eural **N**etworks
Fast RCNN	**Fast R**egions with **C**onvolutional **N**eural **N**etworks
Faster RCNN	**Faster R**egions with **C**onvolutional **N**eural **N**etworks
YOLO	**Y**ou **O**nly **L**ook **O**nce
YOLOv2	**Y**ou **O**nly **L**ook **O**nce **v**ersion **2**
SSD	**S**ingle **S**hot Multibox **D**etector
SVM	**S**upport **V**ector **M**achine
HOG	**H**istograms of **O**riented **G**radients for Human Detection
SIFT	**S**cale **I**nvariant **F**eature **T**ransform
CNN	**C**onvolutional **N**eural **N**etwork
ILSVRC	**I**mageNet **L**arge **S**cale **V**isual **R**ecognition **C**hallenge
RPN	**R**egion **P**roposal **N**etwork
PASCAL VOC	**P**attern **A**nalysis, **S**tatiscal **M**odelling and **C**omputational **L**earning **V**isual **O**bject **C**lasses
DPM	**D**eformable **P**art **M**odel
SGD	**S**tochastic **G**radient **D**escent
AP	**A**verage **P**recision
mAP	**m**ean **A**verage **P**recision
IoU	**I**ntersection **o**ver **U**nion
CUDA	**C**ompute **U**nified **D**evice **A**rchitecture
GPU	**G**raphical **P**rocessing **U**nit
CPU	**C**entral **P**rocessing **U**nit
NMS	**N**on-**m**aximum **s**uppression
ADAM	**ADA**ptive **M**oment Estimation
DSR	**D**esign **S**cience **R**esearch
API	**A**pplication **P**rocessing **I**nterface
(RoI)	**R**egion **o**f **I**nterest
DOTA	Large-scale **D**ataset for **O**bject De**T**ection in **A**erial Images

Chapter 1

Introduction

1.1 Background

Object Detection is a field of study concerned with machines being able to identify objects within images accurately and quickly in real time [1]. With that in mind it is worth noting that there is a considerable body of work that has been done in this field. To further substantiate this, papers such as Papageorgiou and Poggio [2], and Schneiderman and Kanade [3], used various implementations and methodologies in regards to the detection of humans. One of these methodologies (as an example), included the use of a polynomial support vector machine (SVM) which used Haar wavelet descriptors as an input [4].

Girshick et al. [5], note that for a significant amount of time that various objection detection implementations were derived from block-wise orientation histograms such as the Histograms of Oriented Gradients for Human Detection (or HOG) and the Scale Invariant Features Transform (or SIFT) [5]. It is generally accepted that HOG and SIFT are roughly associated with the visual neurological pathway within the first cortical area, but it is also noted that visual recognition takes place through multi-stage processes, which are hierarchical in nature and as such the HOG and SIFT fall short of this [5]. In order to address such shortcomings, a "neocognitron" was conceived by Fukushima which would replicate a neurological structure in terms of how it visually recognises patterns [6]. In light of this, the neocognitron lacked the functionality for supervised learning and as such work by Lecun et al. [7], extended the capabilities of the neocognitron by supplementing it with the use of a

stochastic gradient descent via back-propagation, in order to train a convolutional neural network (CNN) [5].

CNN's were once considered a flagship tool within object detection, but subsequently it fell out of favour due to the popular rise of support vector machines [5]. This trend subsequently reversed when in 2012, Krizhevsky, Sutskever, and Hinton [8], managed to produce a high classification accuracy in regards to the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [5]. Ultimately such results lead to a pertinent question being asked in which, to what extent can the results from image classification, be used to generalize object detection [5]. Girshick et al. [5], address this issue by illustrating that a CNN implementation can ultimately generate higher results in regards to object detection, in comparison to other methodologies/implementations that are derived from simpler HOG-like features [5]. The aforementioned CNN implementation uses a multi-pipeline approach in order to generate accurate results in regards to object detection, as this would require the localization of objects within the image. In order to facilitate such an implementation, the localization of objects could be framed as a regression problem, but another body work [9], illustrates that this approach would not be optimal [5]. In light of this the "recognition using regions" paradigm is ultimately considered and used when implementing the CNN. As such this model is subsequently dubbed the RCNN (Regions with CNN features) [5].

Object detection is generally considered to be a difficult task to complete due to the nature of the problem it faces, as it requires an array of complex methodologies in order to answer its' given task [10]. In response to this complexity, multi-stage pipelines are incorporated in order to train models and as such these pipelines are often slow and time-consuming as well as unrefined [10]. As such the RCNN implementation generates numerous regions (dubbed candidate regions) in order to produce the appropriate localizations within an image [10]. However with this in mind these candidate regions need to be processed and subsequently refined, in order to generate accurate localizations. To remedy these shortcomings, a model that can both classify object proposals as well as refine localizations, is constructed. This CNN model is a version of the RCNN with the ultimate goal of accurately detecting objects in a timely fashion and as such it is dubbed the Fast Region-based Convolutional Network method (Fast RCNN) [10].

A notable drawback of the Fast RCNN as well as the RCNN is that these implementations use a selective search algorithm in order to generate their candidate/proposed regions [11]. In light of this, these networks are negatively impacted at test time and as such in order to address such an issue, the Faster Region-based Convolutional Network (Faster RCNN) was developed [11]. The Faster RCNN is ultimately a culmination of two different modules (the Region Proposal Network (RPN) as well as an object detector network, the Fast RCNN) being integrated into one unified detection model. In order to improve upon performance at test time the Faster RCNN utilizes the RPN in such a manner so as to share its' convolutional layers with the Fast RCNN. This ultimately leads to an improvement in performance at test time since the computational cost for computing proposals will be marginal [11].

The aforementioned RCNN networks (namely the RCNN, Fast RCNN, Faster RCNN) are considered to be multi-pipeline implementations. As such they are considered to be two stage object detectors as they first generate candidate regions and then they subsequently classify/refine the location of the predictions that they make [12]. In order to subsequently improve the shortcomings and overall performance of two stage detectors, single stage detectors are ultimately constructed [12].

The first single stage detector to consider would be the You Only Look Once (YOLO) object detector, which is trained through the use of full images [12]. With this in mind YOLO is characterised as being extremely fast in comparison to the aforementioned two stage detectors. In light of its fast detection YOLO also outperforms other real time systems in terms of accuracy. YOLO is able to achieve this through the use of a single system that frames object detection as a regression problem [12].

Another single stage detector that ought to be considered would be the Single Shot Multi-detector Box (SSD). The SSD is able to perform both faster and more accurately than YOLO and the aforementioned two stage detectors [13]. This is achieved through the use of different scales that are generated from feature maps of different scales, where the predictions will be explicitly separated in regards to their aspect ratio [13]. Furthermore the implementation of design features aided simple end-to-end training which consequently resulted in a high accuracy on images while also improving the trade-off between speed and accuracy, which usually hinders

performance of object detectors [13].

Taking into account that there is also a considerable amount of work being done on remotely sensed images and how well these detectors perform on them [14], one can ultimately see the fundamental bedrock that has been laid out for this field, with regard to the extensive literature that is available. Object detection, as a result, is an expansive field that is a constant state of change and improvement.

1.2 Research Question (or Problem Statement)

1.2.1 Research Area

The research area of concern with regard to this investigation would be image classification (broadly speaking) and object detection (to be more specific). As previously highlighted, object detection involves the use of machines in order to identify objects within images. With this in mind there are various algorithms and methodologies that can bring this into fruition [1]. The significance of this type research is that it has so many broad implications that could be implemented in a wide array of scenarios and contexts (the list is endless). An example of this being that artificial intelligence and machine learning algorithms are being used for self driving cars [15], medical practices to detect physiological ailments or abnormalities [16].

1.2.2 Research Problem

This study seeks to investigate the differences in performance for four object detectors across five different datasets. These object detectors are RCNN, Faster RCNN, YOLO and SSD. That is, this study will ultimately compare the differences in performance between single stage and two stage detectors. With regard to performance, there are two main factors that are of concern, and that would be the test time performance of each image classifier as well as their accuracy.

Research into how well object detectors perform on remotely sensed datasets, is also of keen interest, as the various methodologies would be required to accurately and quickly find small objects/targets within complex backgrounds, in order to see how well the given object detectors perform on these remotely sensed images.

1.2.3 Research Aims and Objectives

The aim of this investigation is to compare four image classifiers on how they perform with regard to object detection across five image datasets. As such the main objective would be comparing and contrasting the performance of single stage object detectors with two stage object detectors across various datasets. The single stage object detectors would be YOLO and SSD, while the two stage object detector would be RCNN and Faster RCNN.

In order to meet the aforementioned aims of this investigation, the following objectives will need to be met:

- To collect five different image datasets.
  - The first dataset that will be collected and used, would be the PASCAL VOC dataset [17], which has been used extensively for research investigations related to image classification as well as object detection.
  - The second dataset that will be collected and used, would be Udacity self driving car dataset [18], which is comprised of different objects that are typically found on the road.
  - The third and fourth datasets that will be collected and used, would be the Eastern North American 24 dataset [19] as well as the Snapshot Serengeti dataset [20]. These two datasets consist of images involving different species of animals within their respective regions.
  - The fifth dataset that will be collected and used, would be the Large-scale Dataset for Object Detection in Aerial Images [21], which is the dataset that is comprised of remotely sensed images.
- To build and implement the relevant methodology for each of the object detectors.
- To compare the results and performance of each of the image classifiers across the five datasets, one of which is a remotely sensed dataset.

- To interpret and make the necessary inferences based on those results, that is the test time performance as well as the accuracy of the object detectors will be scrutinized.
-

1.3 Limitations of this research investigation

There are several limitations and issues to consider for this research investigation. The two main issues are time constraints and the availability of datasets.

With regard to time constraints, computational power is a significant factor to consider. Algorithms are often implemented using powerful machines which would not be available when practically conducting this investigation. This will inevitably affect the results that would be generated. Datasets are often riddled with missing information, be it both annotations or images and as such those chosen to be used will need to be considered carefully.

Chapter 2

Literature Review

This section will focus on the literature that has already explored this given research area.

2.1 Regions with Convolutional Neural Network(RCNN)

RCNNs' were first introduced by Girshick et al. [5]. Within this paper the authors sought to answer "To what extent do the CNN classification results on ImageNet generalize to object detection results on the PASCAL VOC Challenge?" [5].

It follows that this paper shows (the first one to do so) that it is possible to bridge/unify the fundamental aspects of image classification and objection detection by showing that a CNN has the ability to perform substantially higher with regard to object detection on the PASCAL VOC in comparison to models premised on simpler HOG-like features [5]. In order to achieve this the authors tackled two primary problems:

1. The localization of objects within images through the employed use of deep learning models [5].
2. The training of, what one would consider to be a high capacity model, while only using a relatively small amount of labelled data for detection [5].

One particular issue that the authors noted is that unlike image classification, object detection may require the localization of many objects within an image. The authors highlighted that in the past two approaches have been proposed and used. The first

approach frames localization as a regression problem. The authors touch upon the fact that this approach, concurrent with theirs, does not perform well [5].

The second approach would be to make use of a sliding window detector. It needs to be noted that CNN's have made use of this particular approach for about two decades where it has been utilized on constrained object categories [5]. Constrained object categories would include objects like pedestrians and faces [5]. CNNs require that high spatial resolution is maintained and as such in order to meet that requirement, the CNNs only consist of two convolutional and pooling layers [5]. It is made clear that upon conducting this investigation, the authors considered utilizing the sliding-window approach [5]. With this in mind, it needs to be noted that there are challenges that are coupled with such a framework. The main challenge with this framework is that units high up within the network (which consists of five convolutional layers), will consist of considerably large receptive fields (that is 195×195 pixels) and strides (that is 32×32 pixels), within the input image. Subsequently it follows that the utilization of the sliding-window paradigm with regard to making precise localizations, will subsequently lead to an open technical challenge [5].

The CNN localization problem is solved through the use of the "recognition using regions" paradigm [5]. The given methodology results in the the production of approximately 2000 class independent regional proposals for the given input image and thus it also follows that the given CNN will subsequently extract (from each region) a feature vector of fixed length [5]. Subsequently it follows that a class specific linear Support Vector Machine (SVM) is used to classify every proposed region [5]. In order to calculate the necessary fixed size CNN input from each regional proposal, irrespective of the region's shape, a simple technique called the affine image warping, is used [5]. It is necessary to note, that the authors name this system which combines region proposals with CNNs the method RCNN: Regions with CNN features [5].

2.1.1 The Architecture of the Model

In order to generate the necessary regional proposals the authors use a selective search algorithm to do so. This algorithm is employed in order to have a controlled

basis of comparison with work that was done prior to their particular body of work [5].

Moving onto feature extraction, the authors extract a 4096 dimensional feature vector from every proposed region through the use of a CNN (a Caffe [22] implementation). This particular implementation and its architecture details are described quite thoroughly by Krizhevsky, Sutskever, and Hinton [8], as the authors themselves note. Through this implementation features were computed through the use of forward propagating, a mean subtracted 227×227 RGB image, through five convolutional layers as well as two fully connected layers [5].

It is worth noting that the image data (within a particular region) needs to be converted into a format that is compatible with the CNN, as its framework requires inputs of a fixed 227 pixel size [5]. This is done so that computations for the region proposals can be executed [5]. As such it is noted that since there are many possible transformations with regard to the arbitrary shaped regions, the simplest regions are selected. This selection takes place despite the size/aspect ratio of the candidate regions. In addition to this the entire area within a tight bounding box are warped to the prescribed size, furthermore, before this warping is conducted the tight bounding box is dilated so that the warped size will be equal to p pixels of warped image context around the original box ($p = 16$ was used by the authors) [5].

2.1.2 Test Time Deduction for RCNN

Upon running the given implementation during the testing phase of the model, the selective search algorithm was run on a given set of images whereby 2000 regional proposals were extracted [5]. In order for the model to compute the necessary features, each proposal is subsequently warped and then subject to forward propagation through the CNN. Subsequently it follows that an SVM is employed to score each extracted feature vector for every relevant category. It needs to be noted that the SVM trained for that class is used to generate that particular score [5]. A greedy non-maximum suppression was then applied (for each class independently) in order to select candidate regions based on their intersection over union (IoU) overlap with respect to the scored regions in an image [5]. As such regions with a higher

score than the given threshold would be selected and those with a lower score than the threshold would be rejected [5].

2.1.3 Run-time Analysis for RCNN

It is necessary to note that there are two elements which make object detection better and by extension more efficient. The first being that across all parameters, all CNN parameters will be shared. Secondly, the CNN will compute feature vectors that have a low dimensionality. Other approaches and systems (such as the UVA system [23]) have feature vectors of a greater dimensionality [5]. The subsequent result of these two features,will be that the time used to perform the necessary computations for the relevant region proposals as well as their given features, will be amortized over all classes [5]. It is worth noting that only the category specific computations will be dot products between the features, SVM weights as well as the non-maximum suppression [5]. As such when practicalities are kept in mind, all dot products for a given image, will subsequently be batched into a single matrix product [5]. As such the feature matrix is usually considered to be 2000×4096 and the SVM weight matrices $4096 \times N$, where N is considered to be the number of categories [5].

As a result the RCNN has the ability to perform matrix multiplication under ten seconds despite there being 100K classes and this is due to the fact that RCNN's are able to scale to thousands of object classes without the use of approximate techniques [5]. Several efficiencies for RCNN are highlighted when compared to other bodies of work. Most notably when compared to the UVA system [23] which has high dimensional features, the UVA would require 134 gigabytes of memory for 100k linear predictors whereas the RCNN would require 1.5 gigabytes thanks to its low dimensional features. In addition to this, it is highlighted that scalable detection systems which make use of DPMs and hashing, report a mean average precision of 16% on VOC 2007 [17] that is coupled with a run time of approximately 300 seconds where as the RCNN according has a run time of approximately one minute and a mean average precision of 59% [5].

Supervised pre-Training

Supervised pre-training is employed when implementing a RCNN. The authors made use of the ILSVRC 2012 dataset [24] in conjunction with image level annotation in order to discriminatively pre-train their CNN. The pre-training of the model that takes places is performed by the Caffe CNN library [22], [5].

Domain Fine Tuning of the RCNN

In order for the adaptation for the CNN to take place, the stochastic gradient(SGD) training of the CNN parameters, utilized the warped region proposals [5]. The authors [5], largely used an unchanged CNN architecture despite the fact that they replaced the CNN's ImageNet specific 1000-way classification layer with a randomly initialized $(N + 1)$-way classification layer [5].

With regard to the two datasets that the authors use, the PASCAL VOC dataset [17] has $N = 20$, while ILSVRC 2013 [24] has $N = 200$. With respect to the region proposals, the authors considered regions with an IoU overlap of ≥ 0.5 as having a positive ground truth box for that box's class otherwise they are considered to be negative [5]. The authors [5], implemented a training rate of of 0.001 for the SGD so as to allow for fine-tuning in order to make progress that would not hinder initialization. Within each SGD iteration, 32 positive windows were sampled (across all classes) and in addition to this 96 background windows, so that mini batch of size 128 would be constructed. Sampling in favour of positive windows was biased due to their extreme rarity in comparison to backgrounds [5].

Object Category Classifiers of the RCNN.

There are a few drawbacks that come with region proposals. One being how to deal with regions that partially overlap objects. By adjusting the IoU threshold from 0.5 to 0.3 one is able to deal with such a drawback. Girshick et al. [5], illustrates this when conducting their investigation and as such, is able to illustrate that they were able to increase the models performance (mean Average Precision (mAP)) by five points. It subsequently follows that when features are extracted and the application of training labels is implemented, then the optimization of one linear SVM per class takes place. Setting the IoU to zero would decrease the mAP by four points rather

than improving it [5].

The figure presented below, figure 2.1, denotes the functionality of the RCNN as described in the aforementioned text.

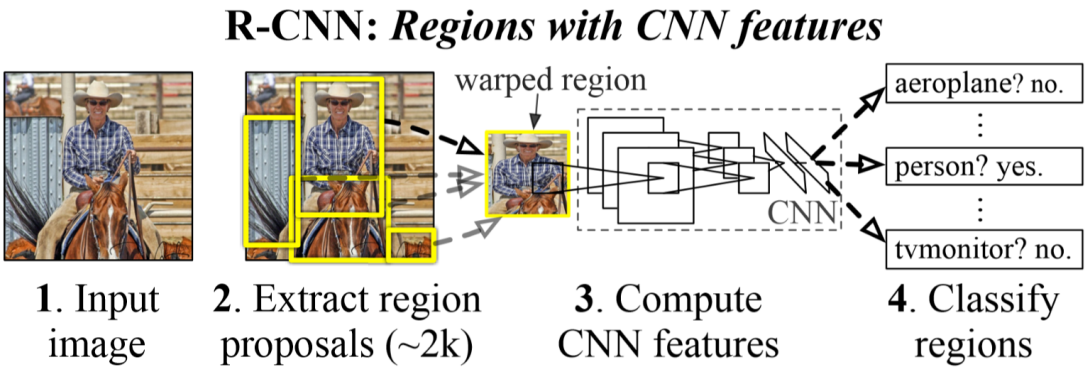


FIGURE 2.1: This figure illustrates the functionality of the RCNN [5].

2.2 Faster RCNN

It is necessary to highlight that RCNNs are executed through the use of CPUs and that the authors, Ren et al. [11], make note of the fact that implementing the region proposals on the GPU will accelerate the computations being made in this regard [11].

In order to overcome some of the drawbacks presented by the RCNN and the subsequent Fast RCNN's, Ren et al. [11] created a network (Faster RCNN) that would address those disadvantages. This improvised network consists of two modules; one being a fully convolutional neural network which generates the proposition of regions (Regional Proposal Network) and a second module (Fast RCNN detector) which then utilises proposed regions [11]. As such this network is considered to be a singular consolidated object detector network. Hence in a rather simple terms the Regional Proposal Network (RPN) tells the Fast RCNN what is of particular interest [11].

One needs to note that an RPN, as its input, takes an image and then subsequently gives as an output, a collection of object proposals which are rectangular in nature, and as such every proposed object will subsequently have a corresponding score (objectness score) that indicates how confident the model is at identifying said objects [11]. The overall objective was to essentially have the computation shared with a Fast RCNN object detection network, as they modelled this process with a full CNN and as such the underlying assumption that was made is that the RPN and the Fast RCNN were sharing a common list of convolutional layers [11].

As noted before the Faster RCNN is comprised of two modules (which are the RPN as well as the Fast RCNN). As such the Faster RCNN is considered to be a unified model where the RPN proposes regions of interest and subsequently tells the Fast RCNN where to to look and which regions to use in order to perform object detection [11].

2.2.1 The Region Proposal Network

The primary responsibility of the RPN network is for it to receive and take in an entire image as an input and as a result it gives as an output a set of rectangular object proposals which are all coupled with a probability score (objectness score) that reflects whether an object lies within the given proposal [11]. It follows that the RPN also has the added feature and responsibility of sharing its subsequent computations with the Fast RCNN since both networks use a common list of convolutional layers. In order for the RPN to generate the necessary object proposals the final shared convolutional layer subsequently slides, what is considered, to be a small network over the convolutional feature map output [11]. A $n \times n$ spatial map is subsequently employed and used as an input for the convolutional feature map [11]. A lowered dimensionality is required for the given spatial window as the generated feature map (of lowered dimensionality) will be passed into two related fully connected layers, which would be a box classification and regression layer [11]. The framework of these networks is subsequently implemented with an $n \times n$ convolutional layer which is subsequently accompanied by 1×1 box classification and regression layers [11].

A basic representation of the Faster RCNN model is given by figure 2.2.

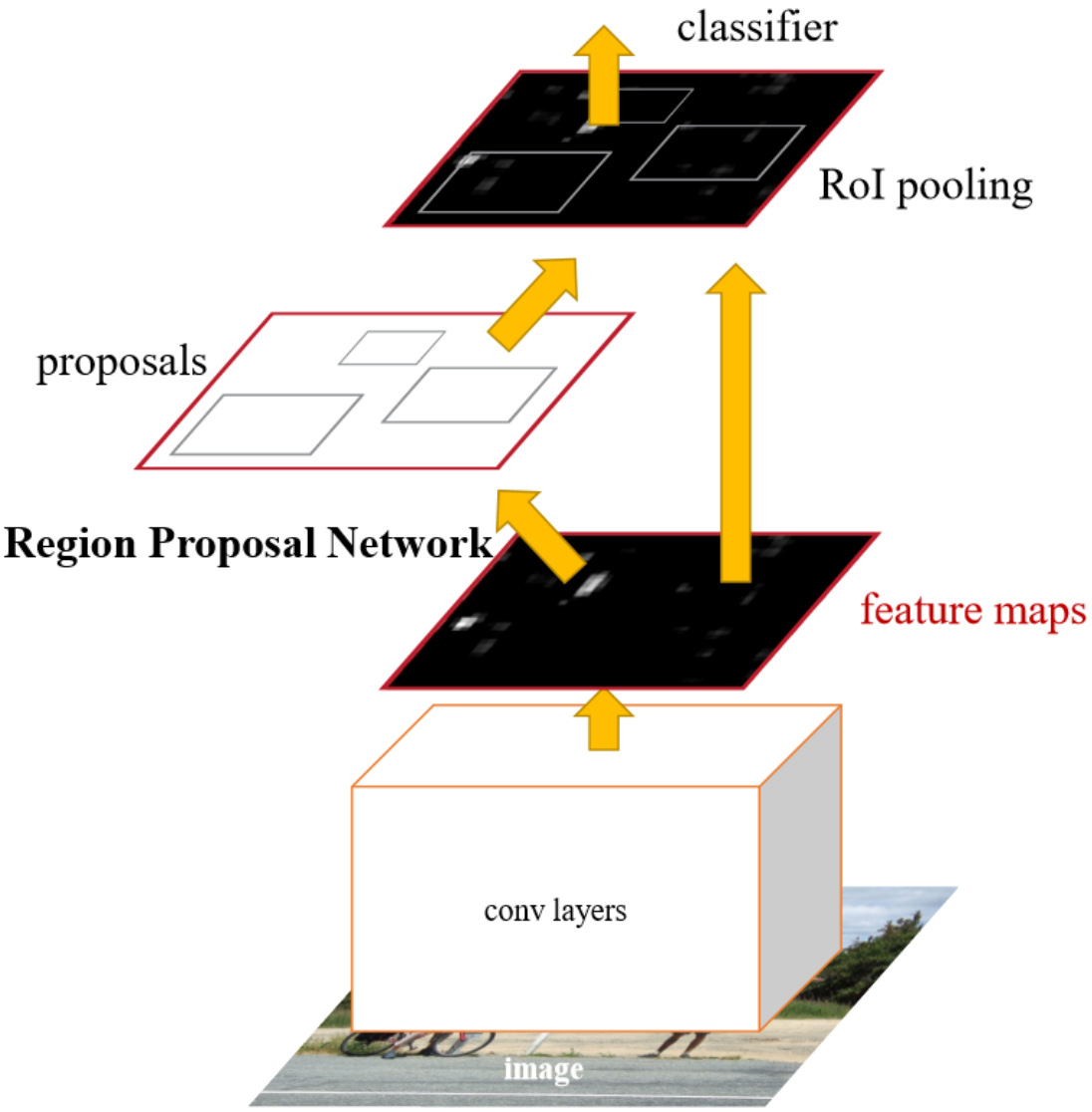


FIGURE 2.2: The figure illustrates a basic representation of the Faster RCNN implementation [11].

Generating Anchors

The number of possible proposals at the given location of each sliding window (denoted by k) will subsequently be predicted for the given number of region proposals [11]. As such it is necessary to note that the k proposals are parametrized in relation

to k reference boxes and as such they are referred to as anchors [11]. In addition to this the anchors are centred at the relevant sliding windows whereby they are associated with the relevant aspect and scale ratios [11].

Translation-Invariant

The translation invariant property is considered to be a very important property. To elaborate further on this, if an object within an image is translated then it follows that the proposal should be translated as well and as such predictions for each object location should be feasible and performed by the same function. Another key feature of this property is that it results in the reduction of size for the model [11]. When comparing the given Faster RCNN approach to a multi-box approach in order to illustrate this phenomenon, the multi-box approach makes use of k-means in order to generate 800 anchors which do not have the given property, i.e they are not translation invariant [11]. Furthermore it needs to be noted that the multi-box model has a $(4+1) \times 800$ dimensional fully connected output layer, whereas Faster RCNN model has a $(4+2) \times 9$ dimensional convolutional output layer [11]. This clearly illustrates that the model Ren et al. [11] developed has an output layer of 2.8×104 parameters, in comparison to the multi-box's output layer of 6.1×106 parameters. This clearly shows that the output layer has substantially fewer parameters when compared to the multi-box [11].

Multi-Scale Anchors

Subsequently it follows that the generation of multiple scale predictions is employed and used to address issues surrounding the use of different scales as well as aspect ratios [11]. There are several methodologies and implementations that can be used to address this particular shortcoming. The first methodology (which is considered to be time consuming) is dubbed image or feature pyramids which involves the resizing of images at different scales so that the computations of the relevant feature includes the resized images at different scales[11]. The second methodology employs the use of sliding windows which are equipped with different aspect/scale ratios and are subsequently utilized on the relevant feature maps. The second methodology is employed to address the aspect surrounding multiple

scales. Then it subsequently follows that it can be considered and dubbed a "pyramid of filters". Both approaches are usually implemented jointly in order to be more cost efficient [11]. It is necessary to highlight that the Faster RCNN is a methodology which utilizes regression and classification bounding boxes in relation to the given anchor boxes which are composed of a single scale and as such they result in the use of a uniformly sized filter [11]. As such in order to resolve this dilemma, a multi-scale paradigm derived from anchors is applied and as such convolutional features are subsequently computed and applied at a single scale for a given image [11].The framework and overall design of the multi-scale anchors are considered to be a key component with regard to how features are shared since they do not carry any extra costs for addressing scales [11].

Loss Function for Training RPNs

Each anchor is allocated two possible class labels (object/not an object). An anchor has to meet any one of two conditions in order to be allocated a positive label [11]:

- An anchor must have the greatest IoU overlap with a given ground truth box [11].
- An anchor must have an IoU overlap greater than 0.7 with any of the ground truth boxes [11].

Keeping the aforementioned conditions in mind, it is necessary to note that a single ground truth box has the ability to allocate positive labels to several anchors [11]. Furthermore the second condition is usually adequate enough in determining positive samples and the first condition will only be applicable in the rare event that the second condition does not find a positive sample [11]. Anchors are given negative labels if they have an IoU overlap which is less than 0.3 for all ground truth boxes [11]. In addition to this anchors that have neither a positive or negative label make no substantive contribution when training the model. As such the aforementioned conditions play a substantive role in minimising the objective function following the multi task loss in the Fast RCNN [11]. It follows that the given image will have

the following loss function [11]:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*) \tag{2.1}$$

It is necessary to define and elaborate further on the variables used in the loss function and as such it follows that [11]:

- i denotes the index within a mini batch and as such p_i denotes the predicted probability of whether or not, anchor i is an object [11].
- p_i^* denotes the ground truth label and as such , an anchor is positive if $p_i^* = 1$ and negative if $p_i^* = 0$ [11].
- The four parametrized coordinates of the predicted bounding box are denoted by the vector t_i [11].
- Subsequently t_i^* denotes the ground-truth box associated with a positive anchor [11].
- L_{cls} denotes the classification loss which is a over two classes (object/not an object) [11].
- $L_{reg}(t_i, t_i^*) = R((t_i - t_i^*))$ is employed for use for the regression loss, and as such it follows that, R is the robust loss function (smooth L_1) [11].
- $p_i^* L_{reg}$ denotes the fact that the regression loss is either enabled for $(p_i^* = 1)$ otherwise it is not enabled for negative anchors $(p_i^* = 0)$ [11].
- The classification layer consists of outputs $\{p_i\}$ whereas the regression layers consists of $\{t_i\}$ [11].
- $\{p_i\}$ and $\{t_i\}$ are weighted by the balancing parameter λ [11].
- It is worth noting that N_{cls} and N_{reg} also normalize $\{p_i\}$ and $\{t_i\}$ [11].

According to the implementation presented in [11] the mini-batch size as well as the number of anchor locations are both respectively employed and used to normalize the classification and regression terms respectively in equation 2.1 [11]. It is prescribed that $\lambda = 10$ so that the classification/regression terms are equally

weighted (approximately so). It is worth noting that the results of the experiments conducted in [11] are insensitive to the various values of λ, furthermore it is subsequently highlighted that the that the normalization of the classification/regression terms is not necessary and it is advised that it should (or could) be simplified. The parametrizations of the 4 coordinates (following the paper [5]) were adapted for bounding box regressions [11]:

$$\begin{aligned} t_x = \frac{(x-x_a)}{w_a}, \quad t_y = \frac{(y-y_a)}{h_a} \\ t_w = \log(\frac{w}{w_a}), \quad t_h = \log(\frac{h}{h_a}), \\ t_x^* = \frac{(x^*-x_a)}{w_a}, \quad t_y^* = \frac{(y^*-y_a)}{h_a}, \\ t_w^* = \log(\frac{w^*}{w_a}), \quad t_h^* = \log(\frac{h^*}{h_a}), \end{aligned} \tag{2.2}$$

It is necessary to note that the box's centre coordinates height as well as its width are denoted by x, y, w, h. The predicted box, anchor box as well the ground-truth box are denoted by x, x_a and x^* [11]. This is generally considered to be a bounding-box regression from an anchor box to a nearby ground-truth box [11]. The authors (Ren et al. [11]) used a different method in order to achieve bounding regression which is done by RoI (Regions of Interest) based methods ([5] and [10]) which were mentioned in the previous chapters. Subsequently it follows that in this given formulation the features used for regression will have the same spatial size (3×3) on the feature maps [11].

Training Hhase for the RPN

Training the RPN would require the use of end-to-end backpropagation as well as the stochastic gradient descent (SGD) [11]. The RPN model is further trained through the use of a sampling methodology that is centred around the image, this is a technique that was also employed for the Fast RCNN [10]. It is worth noting that an image generates a given number of mini batches which are comprised of several negative as well as positive anchor examples [11]. As such the optimization for the given loss functions and all their relevant anchors is a possibility, despite the fact that a bias will exist for the dominant negative anchor examples [11]. With this in mind the arbitrary sampling of 256 anchors within an image is employed so that the computations of the loss function for a given mini-batch are generated in order

to address this bias. As such a ratio of 1 : 1 will subsequently come into fruition for the sampled negative and positive anchor examples [11].

2.2.2 Exchanging Features between the Fast RCNN as well as the RCNN

This section will now deal with what needs to be considered with regards to how the Fast RCNN will utilize the aforementioned proposals. Since the RPN and Fast RCNN are trained separately, as such their convolutions will be subjected to different training regiments [11]. As a result, out of necessity, rather than producing two independent learning networks, a technique is developed that would allow for the sharing of convolutional layers between the networks. There are three solutions in which the training networks have their features shared [11].

- The first is called Alternative training [11].
  - Alternative training entails initially training the RPN, then employing those generated proposals when subsequently training the RPN [11].
  - As a result a the network tuned by the Fast RCNN will subsequently be used to initialize the RPN and as such this process is reiterative [11].
  - The aforementioned solution is utilized by the authors (Ren et al. [11]) when conducting their experiments [11].
- The second solution is called the Approximate joint training and it entails merging the RPN and Fast RCNN networks during training [11].
  - Region proposals are generated for every SGD iteration that takes places during a forward pass [11].
  - When training the Fast RCNN detector, the aforementioned region proposals are subjected to the the same conditions as the fixed, precomputed proposals [11].
  - The computed losses for both networks (RPN/Fast RCNN) are combined despite a backward propagation occurring as per the norm [11].

  - The given solution is considered easy to implement but it is worth noting that it does not take into account certain network responses. More specifically this would be the derivatives of the proposal boxes coordinates [11].
  - Despite the fact the Approximate joint training yields similar results to the Alternative training it generates quicker run time speeds [11].
- The last solution would be the Non-approximate joint training [11].
  - The RPN, predicts the necessary bounding boxes and as such they are also the functions of the input [11].

4-Step Alternating Training.

The adoption of a step training scheme results in alternating optimization because the model has the ability to learn features [11]:

- Step 1: Training for the RPN takes place as previously mentioned before. Subsequently it follows that an ImageNet pre-trained model will be used to initialize the network and as such it subsequently follows that the network is subjected to fine tuning for the region proposal task that takes place end-to-end [11].
- Step 2: The proposals generated in step 1 will be employed by the Fast RCNN so that a different network is subsequently trained [11].
- Step 3: The detector network is initialized first so that the RPN is subsequently trained accordingly, which will result in fine tuning the layers belonging to the RPN as well as fixing the shared convolutional layers [11].
  - Step 3 subsequently results in the two networks sharing convolutional layers [11].
- Step 4: Two processes are undertaken during this step in order to generate a singular network [11].
  - It is prescribed that the shared convolutional layers are kept to be uniform [11].

– It is also prescribed that the layers, belonging only to the Fast RCNN, are fine tuned [11].

Implementation Details

The RPN and Fast RCNN networks are implemented and executed on images comprised of a singular scale [11]. The shorter side of the images are rescaled to $s = 600$ pixels. It is worth noting that the multi-scale feature extraction has a likely chance of increasing the models' accuracy however the added benefit of this does not significantly outweigh the reduction run-times for the model [11].

It is worth noting that the runtime can be reduced or improved upon if the relevant models do not utilize image pyramids/filter pyramids when making predictions for regions of multiple scales [11]. As such it follows that the given implementation has the ability to make predictions which are greater than the underlying amenable field. Upon the commencement of training all cross-boundary anchors are not considered in order to prevent their contribution to the given loss functions [11]. It is worth noting that approximately 20000 anchors can be generated from an image with a size of 1000×600. When training subsequently begins, the number of anchors is further reduced to roughly 6000 per image if the cross-boundary anchors are not considered [11]. With this in mind, it is prescribed that the cross-boundary anchors are ignored as they introduce substantially large and error terms in the objective that would be difficult to correct. This would inevitably lead to the training scheme bing unable to converge [11].

The testing phase of the implementation will require that a fully convolutional RPN is applied to an entire image [11]. As such it subsequently follows that the cross boundary proposal boxes are generated and then attached to the image boundary [11]. It is necessary to highlight that in an effort to reduce the overlapping of RPN proposals (that is their redundancy), a non-maximum suppression strategy is espoused and implemented on the proposed regions in relation to their classification scores [11]. It follows that after NMS the use of the top-N ranked proposal regions for detection, is quite necessary. As as result after training the Fast RCNN using 2000 RPN proposals, it follows that the different numbers will be evaluated at test time [11].

2.3 YOLO

The YOLO implementation is supposed to improve upon the inefficiencies of two stage detectors, this is in part due to the fact that YOLO makes about half the background errors as your Fast Region based CNNs [12]. YOLO is unique in that it brings together the separate components of object detection and consolidates them into a singular network, which results in a network where that uses the features of an entire image so that it is able to predict each bounding box. YOLO is considered to be a unified model which can simultaneously make predictions for each bounding box as well the relevant class probabilities [12]. An exemplification of the

The two main benefits that YOLO has is that it considered to be substantially faster and more accurate than other real time systems [12]. Furthermore YOLO has the ability to learn generalizable denotations of objects, which subsequently results in a model that has a better performance in comparison to other detection models like RCNN when it comes to natural images by rather large margins [12]. It should also be noted that YOLO is less likely to deteriorate and stop functioning when it is used with different and unexpected inputs as well as new domains due to the fact that it is highly generalizable [12].

Despite the obvious advantages of YOLO there are still some disadvantages to this method, namely:

1. YOLO cannot identify some objects accurately, more so small objects [12].
2. Upon analysis conducted by Redmon and Farhadi [1], it is clear that YOLO generates significantly more localization errors than the Fast RCNN [1].
3. Lastly it should be noted that it was found that YOLO has a low recall score when compared to the different RCNN methodologies [1].

The overall design of YOLO is meant to achieve high levels of accuracy while simultaneously achieving real time speeds through the use of end-to-end training. The YOLO implementation will subsequently result in an input image being fractionated into a $S \times S$ grid [12]. Hence the responsibility of detecting the location of the centre of the object is held by each grid cell. As a result it follows that the predictions for the confidence scores as well as B bounding boxes are made for each

grid cell. These confidence scores reflect whether or not the model was able to accurately locate an object within the relevant bounding box. Thus the confidence score can subsequently be defined as $Pr(object)*IOU_{pred}^{truth}$ [12]. Subsequently it follows that if no object lies within the given bounding box then a confidence score of 0 will be given and if the opposite is true than it would be desirable if the confidence score as well as the IoU overlap were equal [12]. It is necessary to state that every bounding box is comprised of five predictions, which are the (x, y) centre coordinates relative to the bounds of the grid cell, the height(h) and width(w), and finally the confidence score [12]. Furthermore the number of B bounding boxes is irrelevant as there will only be one set of class probabilities that will be predicted per grid cell [12]. It follows that at test time, the product between the probabilities as well as the individual box confidence predictions will be implemented as follows [12] :

$$Pr(Class_i|Object)*Pr(Object)*IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth} \tag{2.3}$$

An exemplification and general outline of the underlying process that takes in place in relation to the object detection for YOLO is given by figure 2.3 presented below.

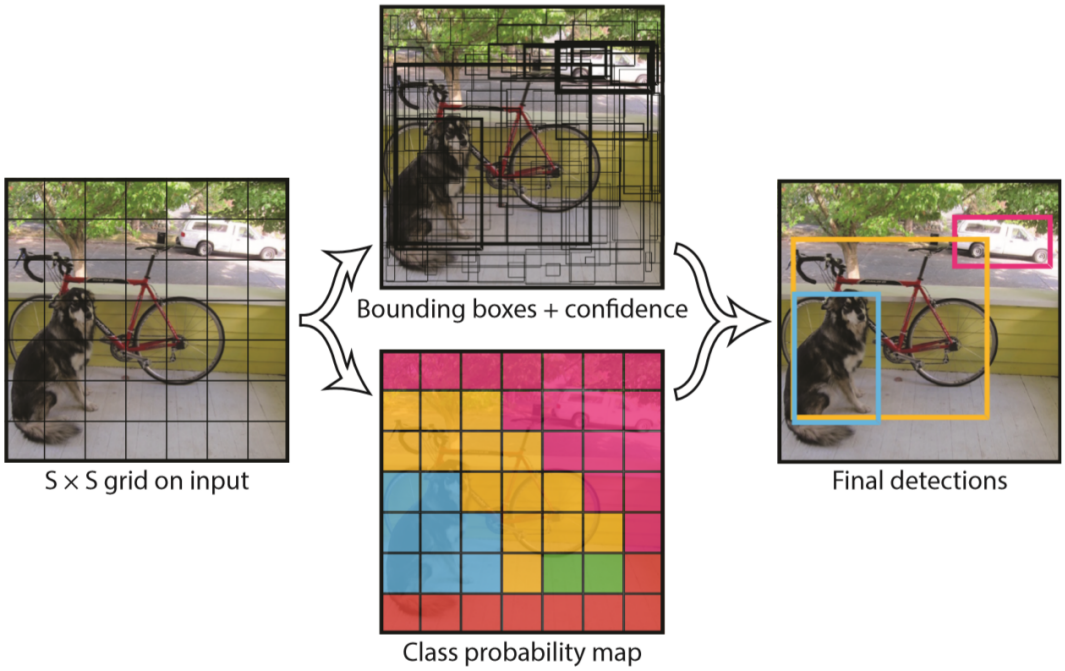


FIGURE 2.3: The figure illustrates the implementation of the YOLO methodology at a basic level [12].

The aforementioned product inevitably leads to the generation of confidence scores for each box (which are class specific). In addition to this, these particular scores are subsequently used to encode the probabilities of how accurately the predicted boxes fit with the objects and whether the correct class lies within the given bounding box [12]. The height and width are also predicted in relation to the whole image. C conditional class probabilities $Pr(Class_i|Object)$ will be predicted for each grid cell [12]. Furthermore these predictions are encoded as an $S \times S \times (B*5 + C)$ tensor, and as a result the grid cell which contains the particular object subsequently conditions these probabilities [12].

It is necessary to note that Redmon and Farhadi [1] present novel concepts to help ameliorate the accuracy of YOLO which subsequently results in a YOLO implementation which would be more accurate but still just as fast. It is subsequently dubbed YOLOv2 [1]. Batch normalization is introduced in order to remove the various implementations of regularization while also improving convergence [1]. A marginal increase in accuracy of YOLO is therefore noted when all of the convolutional layers

are appended with batch normalization. Batch normalization is also noted in aiding the regularization of the model and in addition to this drop-out can be removed from the model without over-fitting [1].

It needs to be noted that a significant portion of image classifiers are pre-trained on ImageNet where input images less than 256×256 are used and operated on. The model as implemented by Redmon et al. [12] would use a resolution of 224×224 to train a classification network. As such the model would subsequently increase the resolution to 448 when detection is being conducted. This subsequently results in a network that alternates between learning to detect objects as it simultaneously adjusts to the latest input resolution [1]. YOLOv2 is different in this regard as it is initially fine tuned at the full 448×448 resolution for the first 10 epochs on ImageNet. This subsequently results in a network where it is given the necessary time to adjust its filters in order for it to work better on high resolution outputs. As such the resulting network on detection is fine tuned further in order to have a resolution classification network whereby an mAP increase of approximately 4% is achieved [1].

As noted earlier YOLO is able to make predictions of the coordinates of bounding boxes directly through the use of fully connected layers on top of the convolutional feature extractor [1]. With this in mind the YOLOv2 will have the fully connected layers removed and the subsequent result will be that, the predictions of the bounding boxes will be made through the use of anchor boxes [1]. Subsequently it follows that the elimination of one pooling layer results in a higher resolution from the output in regards to the networks convolutional layers. What subsequently follows is that the network is shrunk in order to work on the 416 input images rather than 448×448 [1]. The main reason for undertaking this process would be to obtain a feature map with an odd number of locations and as such this would inevitably result in a single centre cell. A notable positive of this is that objects (more so large objects) will a higher tendency of being placed within the centre of an image, so rather than using four locations to predict objects, one location in the centre of an image can be used when making all the relevant predictions [1].

The downsampling of the image reduces an input image by a factor of 32 and

as such an input image of 416 will result in a feature map of 13×13 (downsampling is conducted by YOLO's convolutional layers) [1]. In relation to the anchor boxes, it follows that in order to generate predictions for the anchor boxes, a dissociation between the spatial location and the class prediction paradigm has to occur [1]. It is worth noting that the predicted objects are able to generate the relevant predictions for the IoU overlap score. Redmon and Farhadi [1] note that there will be a small decrease in the accuracy of YOLOv2 due to the use of anchor boxes. With this in mind however YOLO is only able to predict 98 boxes per image whereas YOLOv2, due to the utilization anchor boxes, has the ability to predict more than 1000 boxes for each image [1]. There are notable differences when YOLOv2 is implemented with/without anchor boxes. Most notably there is a substantial improvement in the recall score when YOLOv2 is implemented with anchors $(\text{recall score of } 88\%)$ in comparison to when YOLOv2 is not implemented with anchors is not $(\text{recall score of } 81\%)$ [1]. This inevitably means that despite a small decrease in the mAP the model still has room for improvement due to the increase in recall [1].

There are two main issues that come along with anchor boxes. The first issue is that box dimension are "hand picked" which results in a network that can learn to adjust accordingly [1]. It follows that when these box dimensions are chosen in a more efficient manner, then the network will have better box dimensions to start with and as such the network will be able to learn far much more easily to predict good detections [1]. In order to choose the necessary box dimensions the employment of the k-means clustering algorithm occurs, thus the bounding boxes for the training datasets', will accordingly locate the necessary anchors one would be looking for [1]. It is necessary to note that the conventional k-means algorithm (which employs the use of euclidean distance) generates more errors of larger bounding boxes than it does for smaller bounding boxes [1]. This is in contrast to what is meant to be achieved, which are good IoU scores that would not depend on the size of the box [1]. As a result a distance metric of:

$$d(\text{box}, \text{centroid}) = 1 - IoU(\text{box}, \text{centroid}) \tag{2.4}$$

will be used [1]. K-means is executed for the given number of values of k and subsequently the average IoU will be plotted against the closest centroid so that an

appropriate k value can be prescribed [1].

The second issue that come into play with anchor boxes, would be the model instability which is more of an issue during early iterations [1]. Model instability is noted as being mostly caused by making predictions from the (x, y) locations of the box. It needs to be highlighted that within RPNs, the network predicts the values t_x as well as t_y and calculates the centre co-ordinates (x, y) as [1]:

$$x = (t_x{*}w_a) - x_a$$
$$y = (t_y{*}h_a) - y_a$$

To illustrate this, a prediction of $t_x = 1$ would result in shifting the relevant box to the right by the width of the anchor box, furthermore a predication of $t_x = -1$ would subsequently result in shifting the box to the left by the width of the anchor box [1]. In addition to this, the arbitrary initialization of the network will result in a considerable amount of time being needed to stabilize it, so that sensible offsets are predicted accordingly [1]. In light of this, it would be better for the network to make predictions of the location coordinates in relation to the location of the grid cell rather than predicting offsets [1]. As such the a logistic regression can be used to constrain the networks predictions in order for the bounds of the ground truths to fall between 0 and 1 [1]. In the output feature map 5 bounding boxes will be predicted by the network at each cell. As such the coordinates are predicted by the network for each bounding box, t_x, t_y, t_w, t_h, t_o [1]. It is worth noting that a cell which is offset from the top left corner of the image by (c_x, c_y) and the bounding box prior will have a width and height p_w, p_h, then the predictions corresponds to [1].:

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w \exp t_w$$
$$b_h = p_h \exp t_h$$

The location prediction functions to make the parametrization easier to learn as well as making the network more stable [1]. As such a YOLO model that operates

with dimension clusters as well as with directly predicting the bounding box centre location results in a 5% accuracy improvement compared to a version of the model that only uses anchor boxes [1].

As stated earlier YOLOv2 uses a 13×13 feature map to generate predictions that would also accommodate large objects. With this in mind the YOLOv2 would benefit from finer grained features being used for the localization of smaller objects. The approach that is taken by Redmon and Farhadi [1] in this regard is that a pass-through layer is added which brings in features from an earlier layer at a 26×26 resolution [1]. It is worth noting that the function of this pass-through layer is to concatenate high and low resolution features by placing adjacent features into different channels rather than in spatial locations [1]. As such, this inevitability results in a $26 \times 26 \times 512$ feature map turning into a $13 \times 13 \times 2048$ feature map that can be concatenated with the original features. Redmon and Farhadi [1], note a modest improvement of 1% when running their detector on top of this expanded feature map in order for it have access to fine grained features [1].

2.3.1 Training the model with inputs of multiple scales

The original methodological implementation of YOLO used an input resolution of 448×448 and with the amendment of anchor boxes this is later revised to 416×416 [1]. In contrast YOLOv2 is able to resize input resolutions "on the fly", due to it only using convolutional and pooling layers as it is necessary for the model to be resilient when the model runs images with various sizes, this aspect is subsequently trained into the model [1]. As a result the network is set such that after every few iterations the given implementation downsamples by a factor of 32 from the following multiples of 32 : $\{320, 352, \ldots, 608\}$, where the largest and tiniest samples are 608×608 and 320×320 respectively [1]. The network is resized to those dimensions as a consequence and the model is educated on how to best make predictions fro the various input dimensions. In other words the network learns to make predictions across various resolutions [1]. The authors, Redmon and Farhadi [1], note that at more miniature sizes the network, YOLOv2, is able to run quicker, and as such a trade-off comes into play between speed/accuracy [1]. In addition to this they note that YOLOv2 is able to run as a "cheap" and fairly accurate object detector since at an input resolution of 288×288 it runs at more than 90 frames per second with a

mAP similar to the Fast RCNN [10], whereas at higher resolution of YOLOv2 is still able to operate at real time speeds with a mAP of 78.6 [1].

YOLOv2 is trained through the use of a Darknet-19 framework coupled with the SGD on a ImageNet 1000 dataset [24] Furthermore it needs to be noted that the typical data augmentation strategies are also used during training, namely [1]:

- saturations as well as rotations [12],
- crops [12],
- exposure shifts [12],
- and hues [12].

As noted earlier, initially training on images at 224×224, the network will be subsequently fine tuned to a larger size, 448. It is only necessary to fine tune the network for 10 epochs and subsequently start with a learning rate of 10^{-3} [12].

It is prescribed that the sum-squared error be employed for use due to the fact that it is considered to be rather simple to optimize. As such it is necessary to note that the sum squared error does not line up with the overall objective of maximising the average precision [12]. In addition to this, objects are not contained in a considerable amount of grid cells within every image. As a result the "confidence" scores will push the cells towards zero, which results in the gradient from the cells (which contain objects) being overpowered [12]. With this in mind the training that the model undertakes may diverge during its initial/early phase as the possibility of model instability is raised [12]. In order to address this issue it is prescribed that, for boxes which do not contain objects, the loss is subjected to an increase as well as a decrease for both the predicted bounding box coordinates and the predicted confidence scores [12]. For this outcome to bear fruition it is further given that $\lambda_{coord} = 5$ and $\lambda_{noobj} = 5$ [12]. As noted earlier the sum-squared errors will being about equally weighted errors that are generated by miniature/large boxes [12]. In addition to this the metric also reflects the fact that the large and small boxes do not carry the same consequences as small deviations generated from large boxes tend to carry "less weight" than those from smaller boxes [12]. In order to attend to this, it is prescribed that predictions are made for the square roots of both the height and

width of the boxes rather than their actual height and width [12]. This prescription will subsequently result in the YOLO implementation being able to generate more accurate predictions per grid cell in relation to the multiple bounding boxes [12].

During the training phase of the given YOLO implementation it is required that any one of the bounding box predictors is liable for each object. To further elaborate on this, one predictor will be given the responsibility of generating predictions for a given object based on whether or not the given predictor consists of the highest IoU overlap [12]. This subsequently results in a specialization of the bounding box predictors where the recall scores are subjected to a holistic improvement. This is due to the fact that the each predictor will better suited in predicting their own specialized sizes, aspect ratios and classes of objects [12]. The following multi-loss function is subsequently optimized upon commencement for the training phase of the given YOLO methodology [12]:

$$
\begin{aligned}
&\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[\left(x_i - \widehat{x}_i\right)^2 + \left(y_i - \widehat{y}_i\right)^2 \right] \\
+\lambda_{coord} &\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\widehat{x}_i}\right)^2 + \left(\sqrt{y_i} - \sqrt{\widehat{y}_i}\right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left(C_i - \widehat{C}_i\right)^2 \\
&+\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} \left(C_i - \widehat{C}_i\right)^2 \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} \left(C_i - \widehat{C}_i\right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} \left(p_i(c) - \widehat{p}_i(c)\right)^2
\end{aligned}
\tag{2.5}
$$

it is necessary to note that $\mathbb{1}_i^{obj}$ represents if an object resides in cell i and as such, it represents whether or not the j-th bounding box predictor in cell i, is liable for that prediction [12]. The penalization of a classification error occurs if an object, which lies within the grid cell, is incorrectly classified [12].

It is worth highlighting that YOLO is considered to be a rather fast network in regards to test time. This is due in part to the fact that it is a singular network (as already stated) which would mean that only this singular network is required for evaluation. Spatial diversity within the bounding box predictions is enforced within the grid design. As a result it follows that it is quite apparent that the network is only able to predict one box for each object as well as which particular grid cell an objects falls into. It is also possible for large objects (or objects neighbouring on the border of several cells) to be localized by multiple cells. In order to fix these multiple detections it is prescribed that a non-maximal suppression be used [12].

In conclusion there are several limitations of YOLO that need to be highlighted and discussed. To start things off, YOLO imposes rather strong spatial constraints on bounding box predictions since each grid cell is able to only predict two boxes and as a result will only have one class [12]. Subsequently it follows that this spatial constraint will limit the network in terms of the number of proximal object that it can predict. As such the network will only be able to learn to predict bounding boxes from data, hence it will inevitably struggle to generalize objects in contemporary or rather unusual aspect ratios/configurations [12]. Therefore the network will use rather coarse features in order to predict the bounding boxes due to the fact that its' architecture will have multiple downsampling layers from the input layers [12]. To end things off, the loss function (which approximates detection performance) that is used to train the network, will subsequently end up treating the errors the same in a small bounding box in comparison to a large bounding box [12]. It needs to be noted that a small error within a large box is considered to be rather benign but a small error in a small box will have a much greater effect on the IoU [12]. It needs to emphasised that the primary cause of the aforementioned errors would be the wrong localizations being made [12].

2.4 SSD

The authors, Liu et al. [13], note that the models that were presented earlier have two main shortcomings. The first being that the RCNNs take a considerable amount of time to operate and their quickest model (Faster RCNN) operates at 7 frames per

second while also producing accurate predictions [13]. YOLO on the other hand operates at 45 frames per second but at the cost of accuracy [13].

It worth noting that the SSD model is meant to perform both efficiently (time-wise) and accurately, thereby outperforming both YOLO and the aforementioned 2 stage detectors [13].

SSD is a network framework that uses a single stage to detect objects (that is a single stage detector) as such since this means that classification and localization transpire when a single forward passes through the network. SSD also employs the use of feature extractor network which are also present in the Faster RCNN [13]. SSD also uses a bounding boxes regression that was developed by Szegedy, Toshev, and Erhan [9], and as such the main purpose in that regard is that SSD has no need for regional proposals since different bounding boxes are used then adjust the bounding box for its prediction [13].

2.4.1 Model for the SSD

In order to build and implement the SSD, the use of a framework based on the feed-forward CNN is employed in order for the SSD to be able to generate a fixed sized set of bounding boxes as well as scores that would be necessary for detecting the presence of different object classes within the relevant bounding boxes and as a result, it follows that a non-maximum suppression step will also be utilized so that the final detections can be generated [13].

The initial layers of the single stage multi-box detector architecture stem from a conventional network architecture [13]. A supplementary framework is subsequently appended to the network which will result in it generating detections with a number of key features [13].

The first of these key features necessary for detection, would be multi-scale feature maps. This key feature consists of the addition of convolutional feature layers at the end of the pruned base network [13]. The size of these given layers progressively decreases and as such they facilitate the predicted detections at multiple scales [13]. It follows that the predicted detections that the convolutional model makes will not be the same for each feature layer [13].

The second key feature is the convolutional predictor for detection. Through the use of a set of convolutional filters, the addition of every feature layer will result in the generation of a fixed set of detection predictions [13]. Therefore a feature layer with the given size of $m \times n$ and p channels will result in a $3 \times 3 \times p$ small kernel that generates the relevant score and the particular category, since this would be the fundamental factor needed for the prediction parameters to make a prospective detection [13]. For each m by n location it is prescribed that a kernel be applied to each of the given locations so that an output value can be generated. It follows that the measurement for every feature map location, for the offset values of the bounding box, is subsequently taken in relation with the default boxes' position [13]. Lastly, the third and final key feature would be the aspect ratios as well as the default boxes. It is necessary to note that at the peak of the model, the set of default bounding boxes is related to each of the feature map cells for multiple feature maps [13]. It follows that the default boxes will employ a convolutional manner to arrange the feature maps. Thus the location for every box is subsequently fixed in relation to its given/associated cell [13]. As such it follows that at each feature map cell, offsets are predicted relative to the default box shapes within the cell and the per class scores which indicate whether an object lies within each of those boxes [13]. More specifically it follows that for each box out of k that lie at a particular location, c class scores will be computed as well as the four offsets relative to the original default box shape [13]. What this inevitably results in is a total of $(c + 4)kmn$ outputs for a $m \times n$ feature map [13].

2.4.2 Training for the SSD

When the training for SSD is undertaken, it is required that the ground truth information are allocated to specific outputs in the fixed set of detector outputs [13]. As such it subsequently follows that the loss function and backpropagation will be applied end-to-end [13]. In addition to this the training of the SSD involves a set of default boxes being chosen alongside the scales of detection, data augmentation and hard negative mining [13].

The authors, Liu et al. [13], note that the training objective for the SSD stems from the multi-box objective and as such it follows that it can be extended into handling several object classes [13]. In addition to this, $x_{ij}^{p} = \{1, 0\}$ is defined as the indicator

for matching the i-th default box to the j-th ground truth box of category p [13]. As such it will follow that $\sum_i x_{ij}^p \geq 1$ for the aforementioned matching strategy [13]. As a result the entire objective loss function is composed of a weighted sum of the localization loss as well as the confidence loss where N is defined as the number of matched default boxes [13].

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g)) \tag{2.6}$$

Subsequently if $N = 0$ then the loss will be equivalent to 0 [13]. The localization loss will thus be a smooth L1 loss between the predicted box (1) as well as the ground truth box(g) parameters [13]. In order to offset the centre (cx, cy) of the default bounding box (d) and for its width (w) and height(h) , will be regressed [13].

$$L_{loc}(x, l, g) = \sum_{i \in Pos}^{max} \sum_{m \in \{cx,cy,w,h\}} x_{ij}^k smooth_{L1}(l_i^m - \widehat{g}_j^m) \tag{2.7}$$

$$\widehat{g}_j^{cx} = \frac{(g_j^{cx} - d_i^{cx})}{d_i^w} \quad \widehat{g}_j^{cy} = \frac{(g_j^{cy} - d_i^{cy})}{d_i^h} \tag{2.8}$$

$$\widehat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right) \quad \widehat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right) \tag{2.9}$$

As such it follows that the confidence loss is the softmax loss over multiple classes confidences(c) [13].

$$L_{conf}(x, c) = -\sum_{i \in Pos}^{N} x_{ij}^p \log(\widehat{c}_i^p) - \sum_{i \in Neg} \log(\widehat{c}_i^0) \quad \text{where } \widehat{c}_i^p = \frac{exp(c_i^p)}{exp(c_i^p)} \tag{2.10}$$

Subsequently it follows that through the use of cross-validation the weight term α will be set to 1 [13].

Since SSD is a single network used for object predictions, the use of feature maps from multiple different layers will result in an image being processed at different sizes and subsequently combining the result afterword, furthermore this all transpires while the parameters are being shared across all object scales [13]. The use of

lower and upper feature maps for detection is motivated due to the fact that the authors, Liu et al. [13], note that the other body of works have established the fact that semantic segmentation can be improved through the use of feature maps at lower layers and the addition of global context pooled from a feature will subsequently assist in smoothing the segmentation results [13].

Within a network the feature maps from different levels, are subsequently associated with the different receptive field sizes. With this mind, it needs to be noted that it is not a requirement for the default boxes and the actual receptive fields to be associated or correspond with one another [13]. The tile of the default boxes is constructed in such a manner so that the feature maps inevitably learn to be responsive to particular scales of the objects. In order to compute the scale of the default boxes for each feature map, m is considered to be the feature maps for prediction, and as such the following computation is given [13]:

$$s_k = s_{min} + \frac{s_{max} - s_{min}}{m - 1}(k - 1), \quad k \in [1, m] \tag{2.11}$$

As such the following definitions are given in regards to the terms defined above where $s_{min} = 0.2$ and $s_{max} = 0.9$. This means that the lowest layer will have a scale of 0.2 and the highest scale will have a scale of 0.9 and as a result all the layers in between will be regularly spaced [13]. In order to compute the width($w_k^a = s_k\sqrt{a_r}$) and the height($h_k^a = \frac{s_k}{\sqrt{a_r}}$) of each default box, it is prescribed that different aspect ratios be imposed for the default boxes and as such they are subsequently represented as $a_r \in \left\{1, 2, 3, \frac{1}{2}, \frac{1}{3}\right\}$ [13]. With regard to the aspect ratio of 1, the result of 6 default boxes per feature map location is given by the addition of a default box with the scale of $s_k^{'} = \sqrt{s_k s_{k+1}}$, furthermore, ($\frac{i+0.5}{|f_k|}$, $\frac{j+0.5}{|f_k|}$, where $|f_k|$ will be the size of the k-th square feature map $i, j \in [0, |f_k|)$, will be set as the centre of each default box [13]. Furthermore in order to have a variety of different scales and aspect ratios from all the locations of the many feature maps, the predictions for all the default boxes with different sales and aspect ratios will be combined [13].

The result after the matching step is that most of the default boxes will be negatives, more so when there is a large number of possible default boxes [13]. Subsequently a considerable imbalance will now exist between the positive and negative examples

[13]. To solve these dilemmas and avoid the employment of all the negative examples, the highest confidence loss will be sorted for each default box and the top ones are chosen so that the ratio at most $3 : 1$ between the negatives and the positives [13]. The authors, Liu et al. [13], found that this will result in faster optimization and a more stable training. The aforementioned step is called hard negative mining [13].

The last training strategy to be employed for the SSD would be the data augmentation strategy. In this regard there are three options that can be employed in order to ensure that the model is holistically more robust to the various input object sizes and shapes, and in addition to this each training image will be arbitrarily sampled [13]. The first option that is available would be to use the entire image as an input. The second option would to arbitrarily sample a patch and the third option would be to sample a patch resulting in the minimum jaccard overlap with the objects is $0.1, 0.3, 0.5, 0.7$ or 0.9 [13]. It is worth noting that the size of each sampled patch is $[0.1, 1]$ in regards to the original size of the image and the aspect ratio is between $\frac{1}{2}$ and 2 [13]. Following this sampling step, each sampled patch is subsequently resized to fixed size and is horizontally flipped with probability 0.5 [13].

Chapter 3

Research Methodology

3.1 Introduction

This section deals with the research method which will be used and applied for this project as well as the Methodology (which involves the Proposed Method, Data, the analysis), Limitations and the Ethical Considerations.

3.1.1 Research Design

A Design Science Research methodology will be used when conducting this particular research project. As such the Design Science Research (DSR) methodology consists of a collection of synthetic and analytical techniques as well as perspectives which are used for executing research objectives/investigations (more so in Computer Science related fields) [25].

The primary functionality of this research design uses the application of prior knowledge in order to help solve as well as address significant issues and problems in an efficient manner [25].

The DSR methodology is compromised of three cycles, namely, relevance, design and rigor. The relevance cycle initiates the DSR methodology into a framework of implementations that sets out research requirements into inputs and also identifies the required criteria for the final assessment of the research outcomes [26]. In relation to the models that were used for this investigation, the relevance cycle is employed when implementing the models and their subsequent algorithms.

The design cycle is described by Simon [27], is the core and heart of every DSR project, and as such this particular cycle is used create various designs and evaluating these different designs against the relevant criteria until such a point that the given design would be satisfactory [27]. In relation to the models that were used for this investigation, the design cycle is employed in relation to the evaluation and performance assessment of the aforementioned object detector models.

The rigor cycle provides the foundational knowledge and methods that were previously established within this research domain and as such the newly generated knowledge from the conducted research will be added as new knowledge to the research community [26]. In relation to the models that were used for this investigation, the rigor cycle is employed in the introductory phase of this investigation, whereby the background and literature review of this research area is given.

3.2 Methodology

3.2.1 Proposed Method

A brief but concise description will be given for necessary methodology that will be carried in order to conduct this investigation. The relevant models will be implemented using python [28], and Application Programming Interfaces such as Keras [29] and the open source library called TensorFlow [30].

Methodology for RCNN

The fundamental steps needed to implement the RCNN model 3 can be summarised as follows [5]:

1. Step 1: Generating region proposals.
    - Through the employment of the selective search algorithm, category-independent region proposals will be generated as these proposals will define the set of candidate object regions [5].
2. Step 2: The creation of CNN features.

- Affine image warping will be used to produce a CNN input, of uniform size, from each candidate region that was generated in step 1 [5].
- Subsequently it will follow that a feature vector of fixed length will be extracted through the employment and use of a pre-trained CNN network [5].

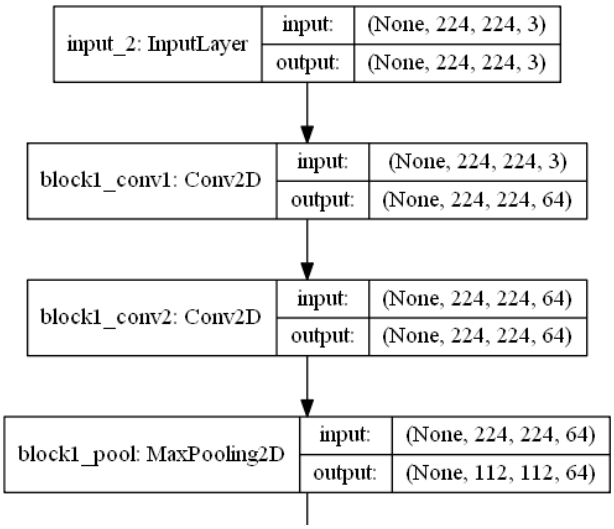
3. Step 3: The Support Vector Machine classifies each region into classes.
    - Each region will then be classified into a class by a set of class specific SVMs [5].

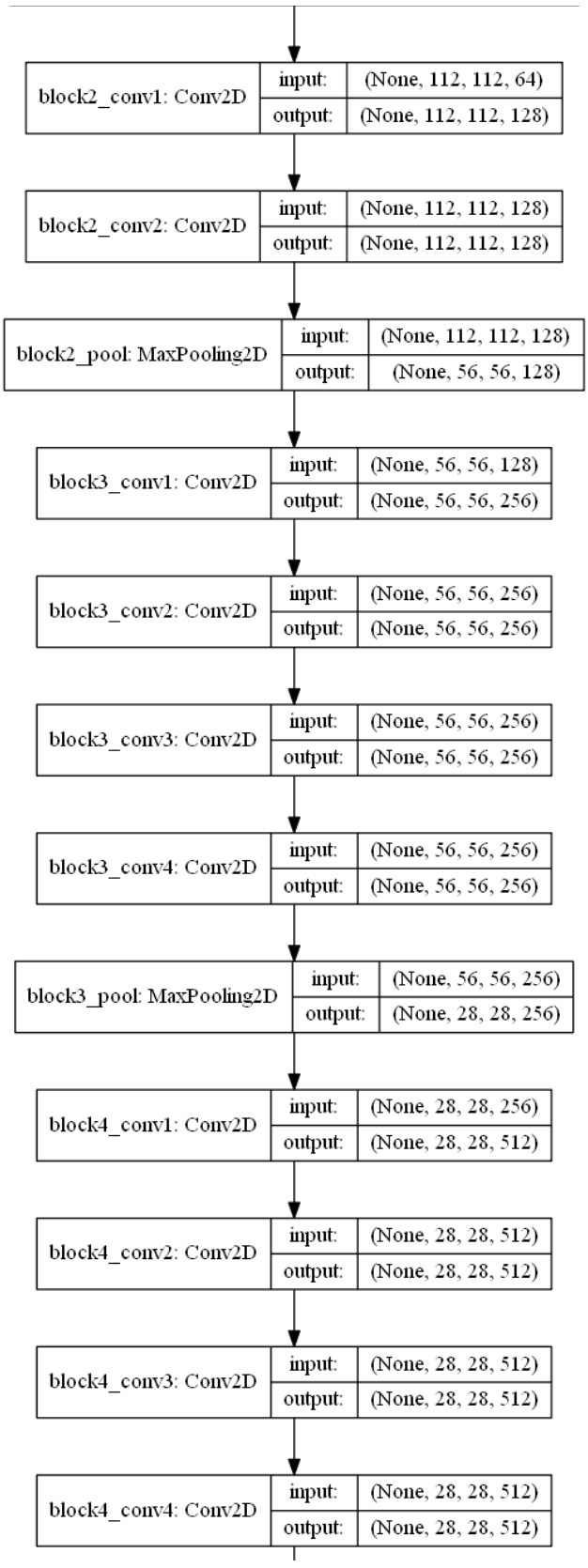
In order to implement the RCNN, a VGGNet [31] will be used to extract the necessary features. VGGNet [31] is quite similar to AlexNet [8] (which was originally used by Girshick et al. [5]) in that it uses 3×3 convolutions. The stark difference between AlexNet [8] and VGGNet [31] is the number of filters being used, as VGGNet [31] has a greater number of filters [8], [31].

VGGNet is also appealing to use in this regard because its' weight configuration is publicly available on application programming interfaces like Keras [29]. It is necessary to note that pre-trained weights from ImageNet [32] will be downloaded and used so as to aid the training of the given implementation.

The last two layers for the VGGNet network will be removed as it will only be used for the extraction of features. This is similar to what was done in Girshick et al. [5] as they removed the last two fully connect layers as well, which subsequently led them to utilize the pool layer [5].

The model summary is presented in the graphic, 3.1, below:


input_2: InputLayer
input:
(None, 224, 224, 3)
output:
(None, 224, 224, 3)
block1_conv1: Conv2D
input:
(None, 224, 224, 3)
output:
(None, 224, 224, 64)
block1_conv2: Conv2D
input:
(None, 224, 224, 64)
output:
(None, 224, 224, 64)
block1_pool: MaxPooling2D
input:
(None, 224, 224, 64)
output:
(None, 112, 112, 64)


block2_conv1: Conv2D
input: (None, 112, 112, 64)
output: (None, 112, 112, 128)
block2_conv2: Conv2D
input: (None, 112, 112, 128)
output: (None, 112, 112, 128)
block2_pool: MaxPooling2D
input: (None, 112, 112, 128)
output: (None, 56, 56, 128)
block3_conv1: Conv2D
input: (None, 56, 56, 128)
output: (None, 56, 56, 256)
block3_conv2: Conv2D
input: (None, 56, 56, 256)
output: (None, 56, 56, 256)
block3_conv3: Conv2D
input: (None, 56, 56, 256)
output: (None, 56, 56, 256)
block3_conv4: Conv2D
input: (None, 56, 56, 256)
output: (None, 56, 56, 256)
block3_pool: MaxPooling2D
input: (None, 56, 56, 256)
output: (None, 28, 28, 256)
block4_conv1: Conv2D
input: (None, 28, 28, 256)
output: (None, 28, 28, 512)
block4_conv2: Conv2D
input: (None, 28, 28, 512)
output: (None, 28, 28, 512)
block4_conv3: Conv2D
input: (None, 28, 28, 512)
output: (None, 28, 28, 512)
block4_conv4: Conv2D
input: (None, 28, 28, 512)
output: (None, 28, 28, 512)

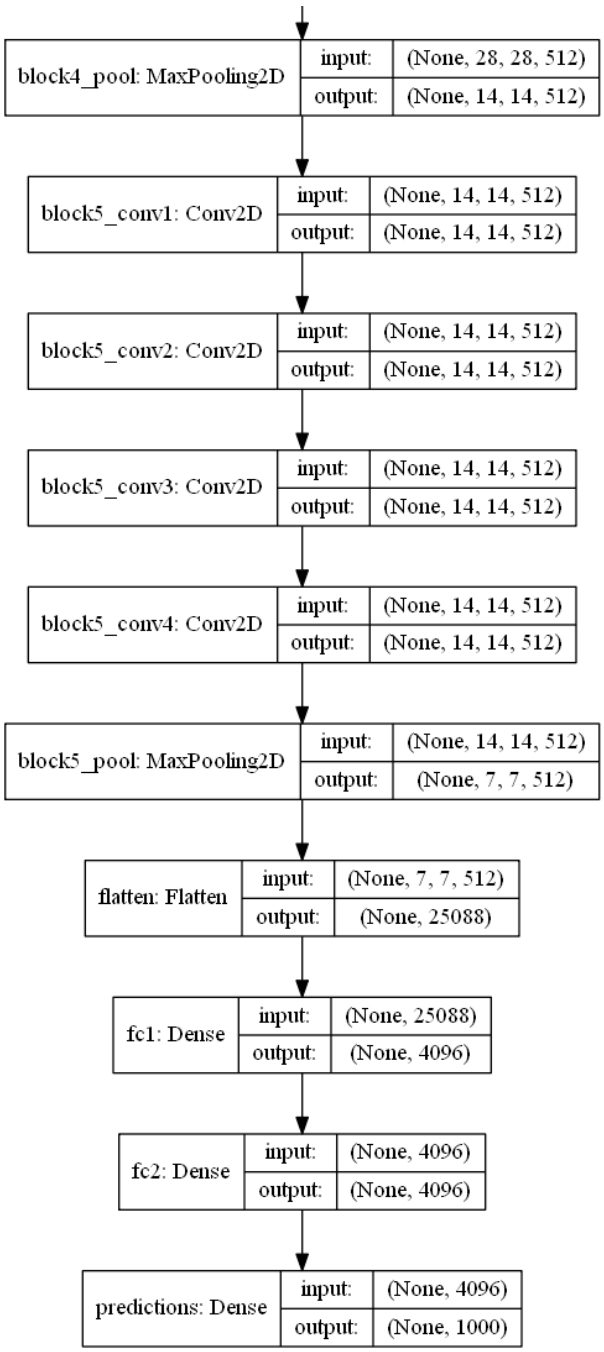


FIGURE 3.1: This figure illustrates the model summary of the VGGNet that was used [29], [31].

It is clear that the image shape should be $(\text{height, width, Nchannel}) = (224, 224, 3)$. As such the search region proposal will not always generate an image with a

height, width $= 224$. The computations for the CNN features for a given region proposal can only be calculated if the image date is subsequently converted into a compatible format. In line with what was highlighted in the literature review, a warping of the region proposals in relation to the CNN's input size, needs to be performed irrespective of the size/aspect ratio of that particular region [5]. As such it follows that once the selection search algorithm has been run on an image approximately 2000 region proposals should be extracted, and thus the warping of each candidate region needs to take place [5].

Finally in order to complete the RCNN model it is necessary to have/create a classifier to identify objects. As stated before, the classifier in this case would be a category specific linear Support Vector Machine that would be trained using the corresponding ground truth label for each candidate region [5]. Regions that contain the relevant object(s) will be considered as positive examples whereas the opposite is true. That is regions that contain no object will be considered to be negative examples. An Intersection over Union overlap threshold ($= 0.3$) will also be used when regions partially overlap objects [5].

Methodology for Faster RCNN

There are several steps that need to be undertaken when implementing the Faster RCNN. To start things off, the VGG16 [31] network is once again employed to be used as a feature extraction module. This network will provide the necessary framework for the two main modules for the Faster RCNN (that is the RPN and Fast RCNN networks) [11]. The VGG16 [31] network layers will be modified where necessary so as to allow the input images to pass through it [11].

Moving onto anchor boxes, they will have two main responsibilities with regard to their implementation [11].

- Anchor boxes will be produced at each feature map location [11].
- As such the appropriate labels as well as the appropriate locations of all the objects will be associated to each anchor [11].
  - The objects within the image will be subsequently allocated to the prescribed anchor box that contains them [11].

  - An anchor must have the greatest IoU overlap with a given ground truth box [11].
  - An anchor must have the an IoU overlap greater than 0.7 with any of the ground truth boxes [11].
  - Anchors are given negative labels if they have an IoU overlap which is less than 0.3 for all ground truth boxes [11].
  - In addition to this anchors that have neither a positive or negative label make no substantive contribution when training the model [11].
- Locations will subsequently be allocated to each anchor box with its relevant ground truth object which has the maximum IoU [11].

Moving onto the RPN network, it has already been noted that the network is comprised of a convolutional module and a regression layer. The regression layers' functionality in this instance, is that it predicts the location of the the relevant box inside the anchor [11]. It follows that in order to generate region proposals the aforementioned network needs to be slid over a convolutional feature map [11]. A $n \times n$ spatial map is subsequently employed and used as an input for the convolutional feature map. It will consequently follow that each sliding window will be mapped to a lower dimensional feature (quite possibly 512 features) [11]. As a result this feature is fed into two related full connected layers which are, as stated before, a box regression and classification layer [11]. The architecture of the Faster RCNN subsequently has a structure of a 3×3 convolutional layer with two related 1×1 convolutional layers [11]. In addition to this the network layers are initialized with a zero mean as well as a 0.01 standard deviation with zeros for base. The Faster RCNN will also be run for 120 epochs with a thousand steps for each epoch. Implementation will subsequently be conducted through the use of the Keras API [29].

It is worth noting that the generated RPN proposals will overlap with each other to a great extent as such it is prescribed that a non-maximum suppression algorithm is adopted so as to address this issue and reduce redundancy as well as maintain accuracy [11].

Subsequently it follows that after the final region proposals have been obtained, the region proposals, ground truth boxes and their relative labels are taken as an input for the Fast RCNN, which is subsequently liable for the predicting the object locations and the relevant class [11]. The RoI pooling layers are subsequently used by the Fast RCNN for each and every proposal suggested by the RPN. The RoI pooling layer is liable for generating feature maps of uniform size, by taking inputs which are not of a fixed size [11]. As such the RoI pooling layer receives the two inputs [11]:

- The first input would be a feature map of uniform size which is retrieved from a convolutional network [11].
- The second input would be the set of RoI's denoted by the matrix, $N \times 5$. The number of regions of interest is denoted by N [11].

The RoI pooling layer is responsible for every region of interest from the input list and as such it takes a section of the input feature map that is associated to it and will subsequently scale it to some pre-defined size [11]. The scaling is subsequently conducted by [11]:

- Each proposed region is fractionated into parts of unifom size [11].
- For each uniform part, it is necessary to determine its greatest value [11].
- Subsequently it follows that these maximum values are replicated into the output buffer [11].

Inevitable a set of rectangles (of various sizes) is obtained and as such the corresponding set of feature maps(of uniform size) can be quickly found. Furthermore the RoI pooling layer is primarily used because it has a comparatively fast processing speed [11]. Following the implementation of the Fast RCNN, once the relevant feature maps are passed through it expected output would be the necessary classification that needs to take place [11].

Lastly the loss function for the Fast RCNN needs to be implemented. Since the Faster RCNN is comprised of two modules, this will result in two loss functions being computed but later added together in order to generate the total loss for the Faster RCNN model [11].

Methodology for YOLO

YOLO employs Anchor boxes in order to localize multiple objects which are in close proximity to one another. As such it is prescribed that it is necessary to predefine two hyper-parameters, which are firstly the number of anchor boxes as well as their shapes [12]. This is done so that multiple objects that in a close proximity to one another will be allocated different anchor boxes. As a result this will be that the more anchor boxes there are the more objects YOLO will be able to detect within close proximity of each other [12].

In order to determine/prescribe the necessary amount of anchor boxes as well as their designated shapes, k-means clustering algorithm is implemented such that it is employed and used with regard to the shape of the relevant bounding box [1]. This is done to prevent an overspecialization of anchor boxes, with regard to predicting the shape of bounding boxes for objects that are a rare data point within the dataset [1].

The feature data that will be used by the k-means clustering will be data that contains the height and width as its features. Subsequently it follows that the height and width of an image will be used to standardize the height as well as the width of the relevant bounding box [1]. This prescription is given because the different image will have a wide array of dimensions (that is height and width) [1].

The k-means clustering will have four primary steps where the number of clusters will be set and the cluster centres will be initialized [1].

1. Step 1: Each item will be allocated to the closest cluster centres. It is worth noting that the IoU will be used to calculate the distance to the cluster centre
2. Step 2: The necessary computations for the cluster centres will take place.
    - The mean/median can be used to bring this about.
3. Step 3: The previous two steps will be repeated up until the two consecutive iterations produce the same cluster centres.

It follows that the k-means clustering is subsequently employed so as to determine the necessary amount of anchor boxes as well as how they will specialize in relation

to the shape of the anchor box. K-means will be run (in order.) for the following hyper parameters: $k \in [2, 10]$ [1].

The use of an Elbow curve or GAP statistics [33], where the main aim in this regard would be to determine the true number of clusters that would be captured, in relation to the slop of the IoU [1].

Input/output encoding of YOLO is another essential process that needs to be undertaken. Input encoding simply requires that an image be read-in and resized to the prescribed shape. With regards to output encoding there are several steps that need to be conducted [1].

- Step 1: The outputs xmin, ymin, xmax, ymax need to be resized [1].
- Step 2: Each object needs to be assigned a corresponding ground truth anchor box [1].
- It has been already established that YOLO needs to use an anchor box to detect multiple objects in a nearby region as such [1]:
  - K-means will be used as describe to determine and prescribe the amount of necessary anchor boxes [1].
  - Each anchor box has its own specialized shape [1].
  - A grid cell that has the greatest IoU overlap as well as the centre and anchor of the object will be coupled with the given object from that particular image [1].
- Step 3: It follows that a rescaling of the unit of bounding box coordinates to the grid cell scales takes place [1].
  - It needs to be noted that YOLO splits the given image into grid cells (of $n \times n$ grid cells) and thus it will subsequently assign image classification and localization algorithms in each of the grid cells [1].
- Step 4: The input and output encodings will be combined and subsequently a batch generator will be produced using the Keras [29] API.

- A Keras generator will be used in this regard to get batches of the input alongside the corresponding output, simultaneously during the training phase of this implementation.
- To illustrate this phenomenon if the model reads in 15 images, 15 corresponding label vectors will be produced as a result.
- According to the authors [1], what will subsequently follow, is that the 15 corresponding label vectors will be fed into the GPU during the training phase.
- For this investigation however usage of the GPU is not possible due to the lack of hardware and software. In light of this Google colab and other online resources, could have been a viable alternative but the internet data requirements would have been a limiting factor as they can be expensive.
- As such the CPU will be used in this regard.

Moving onto the model architecture for YOLO, it is a rather simple design. For the most part it is simply a combination of different layers. Pre-trained weights [1] will be loaded and used for YOLO. It follows that the loss function will be implemented so as to penalize the classification error if an object is present in that grid cell [1]. Subsequently it will follow that the IoU overlap will be computed for each grid cell and anchor pair [1]. For each predicted bounded box the best IoU is calculated irrespective of the ground truth anchor box for each object [1].

Methodology for SSD

The framework for the SSD will based off of the VGG16net [31] and once again implemented using the Keras API [29]. This base network will also be pre-trained on the ILSVRC [24] dataset. During the implementation of this base network it will follow that the original VGG fully connected layers will not be used. Instead similar to what Liu et al. [13] constructed, a set of supplementary convolutional layers will be added so that features can be extracted at multiple scales while simultaneously reducing the size of the input associated to each consecutive layer [13].

The supplementary convolutional layers are meant to also generate key features in order for the network to generate detections. These key features would be the multi-scale feature maps for detection, the default boxes and aspect ratios, the convolutional predictors for detection [13]. Moving onto the necessary steps that will take place when this model is implemented [13].

- The matching strategy is employed so that the ground truth information are assigned to specific outputs in the fixed set of detector outputs [13]. As such it subsequently follows that the loss function and backpropagation will be applied end-to-end [13].
- Hard negative mining takes place due to the fact that there is an expectation, that most of the default boxes will be negative after the matching step [13].
- This is done to address the imbalance between positive and negative training examples so that the model is optimized quicker and the training that takes place is more stable [13].
- The training objective for the SSD will result in an indicator for matching the i-th default box to the j-th ground truth box of category p [13].
- The selection of scales and aspect ratios for the default boxes will take place [13].
  - Since the SSD is a single network used for object predictions [13].
  - The use of feature maps from multiple different layers will result in an image being processed at different sizes and as such the results will be combined [13].
  - Furthermore, this all transpires while the parameters are being shared across all object scales [13].
- Lastly data augmentation can be used so that the model is holistically more robust to the various input object sizes and shapes, and in addition to this each training image will be arbitrarily sampled [13]. There are three main ways one could augment the data and they are as follows [13] :
  - The original image will be used as an input in its entirety [13].

  - Arbitrarily sampling a patch [13].
  - Sampling a patch resulting in the minimum jaccard overlap with the objects is $0.1, 0.3, 0.5, 0.7$ or 0.9 [13].

The SSD model will be subject to a number of training parameters [13].

- The model will be trained for 120 epochs, where each epoch will consist of 1000 training steps.
- The learning rate for the first 80 epochs will be 0.001, followed by 0.0001 for the next 20 epochs and subsequently 0.00001 for the next 20 epochs.
- The optimizer that will be used for the SSD will be the SGD.
  - SGD is employed in this investigation because it has a better generalization performance over adaptive gradient algorithms like ADAM [34].

It is worth noting that the GPU was used in the work conducted by Liu et al. [13], as such for this investigation since the adequate hardware and software specification have not been met a CPU will be used instead. In conclusion it follows that a non-maximum suppression step will subsequently be utilized to make the final detections [13].

3.2.2 Datasets

When conducting an investigation centred around object detection, there are several aspects one has to consider when deciding which dataset(s) to use. Some of these key aspects would include the size, structure, and relevancy of the datasets.

There is a considerable amount of image datasets which one could use for object detection. Their size in terms of image resolution and the overall hard disk memory space they occupy is thus of great consequence. Such features affect the overall performance of the models that one intends to implement, as issues around runtime and accuracy come into play. It is worth noting that the hard disk space that these datasets occupy can range from a couple of gigabytes to more than 200 gigabytes [5].

When considering the relevancy of the image datasets, one has to focus on the main aim of the research investigation. Localizations and the overall performance of the models will be greatly affected and as such performance will vary. As already noted before in the literature review, object sizes within images will not have the same localization accuracy in comparison to larger objects.

Several datasets were used to conduct this research investigation. The first dataset that was utilized would be the PASCAL VOC [17] dataset which contains 20 different image classes [17]. This dataset is used a considerable amount times when conducting investigations centred around object detections.

The second dataset that was used, would be the Udacity self driving car dataset [18]. This dataset four different object classes and subsequently contains over 15000 images and 97942 labels.

A camera snapshot dataset of animals was employed for this research project namely the Eastern North American 24 dataset [19]. This dataset consists of images of different animals from North America in their natural habitat. Hidden cameras were used to obtain these images of animals that were in a reasonable vicinity. The Snapshot Serengeti [20] dataset was also utilized. These snapshot datasets are similar to the Eastern North American 24 dataset, only difference would be the location of where the images were taken and which animals were captured.

With regard to remotely sensed images, the Large-scale Dataset for Object Detection in Aerial Images (DOTA [21]), was employed. The images in this dataset were rather considerably larger than the other datasets and as such preprocessing techniques had to be utilised in order to perform experiments on this dataset. To be more specific the images (and their respective annotations) were split into smaller files and as such this altered the dataset that was used for the given implementations.

3.2.3 Analysis

In this research it would be appropriate to use several tools to analyse and assess the performance of the various methods/implantations. Amongst the several tools one could use the mean Average Precision (mAP) [35] and detection Average Precision

(AP) [36]. The given datasets will be divided into training and testing splits with an 80:20 split. Furthermore the time taken to make the predictions is also of great essence as one of the main objectives of object detection is to have models that are able to make predictions accurately and quickly in real time.

The metrics (AP [36] and mAP [35]), are tools one uses to measure and gauge how accurate object detectors are. These metrics are used due to the fact that object detectors are built to classify and localize objects within an image [37]. In order to make the necessary calculations for the mAP, there are other key sets of information that needs to be accounted for. This information is comprised of the ground truth data and the intersection over union (IoU) overlap [38] as well as the precision and recall [39] scores.

To start things off, the ground truth data for the images contains the necessary annotations needed for the detection and classification to take place. These annotations usually contain information pertaining to the coordinates(x, y), dimensions (height and width) and the class labels of the objects in the image. As a consequence of all this, the ground truth data is subsequently used to generate the ground truth boxes [38]. With this in mind the IoU overlap would be the given ratio from the intersection over union overlap between the predicted boxes that a given object detection model generates and the ground truth boxes. To further elaborate, the overlap between a predicted box and a ground truth box represents the area of intersection for these boxes, while the union overlap simple represents the total cumulative area for these two boxes [38]. Mathematically it can be defined as follows [38]:

$$IoU = \frac{\text{prediction boxes} \cap \text{ground truth boxes}}{\text{prediction boxes} \cup \text{ground truth boxes}} \tag{3.1}$$

or it can be defined as

$$IoU = \frac{\text{Area of overlap}}{\text{Area of Union}} \tag{3.2}$$

As mentioned before there is an IoU threshold that needs to be considered when the model is being executed. This is done so that the True/False positives can be

identified by the model. To further elaborate on this if a threshold of 0.5 is used then any IoU overlap that is greater than 0.5 is considered a True positive otherwise it is considered to be a False positive [39]. In light of this there False negatives also need to be accounted for. False negatives are generated when the model is not able to identify objects within the image [39]. As such it would be possible to compute the correct number of detections that the model is able to generate through the use of a metric called Precision, which is defined as [39]:

$$Precision = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{3.3}$$

Furthermore in order to calculate the average precision of the class of an object the recall score is necessary. In simple terms th recall score is considered to represent how well a model is able to label positive samples [35]. Recall is defined as follows [39]:

$$recall = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negative}} \tag{3.4}$$

Now the average precision is defined as the area under the interpolated precision/recall curve for a given object class [39]. An interpolated precision/recall curve is defined by the subsequent plotting of the recall/precision scores (where the inclusion of lower scored detections is accounted for) [39]. Interpolations are generated for the dips that occur for the given plot, that is every precision is replaced by the highest score of itself as well as the precision scores present at greater uniformly spaced recall levels [39]. The Average Precision [36] can as such be defined at a set as [39]:

$$Average\ Precision = \frac{1}{R} \sum_{r \in \{0, 0.1, \ldots, 1.0\}} p_{interp}(r) \tag{3.5}$$

where R denotes the number of recall levels used and

$$p_{interp}(r) = max_{\widetilde{r}:\widetilde{r} \geq r} p(\widetilde{r}) \tag{3.6}$$

as such the recall level is denoted by r, and $p(\widetilde{r})$ denotes the generated precision score at its corresponding recall $\widetilde{r}$ [36].

In conclusion it follows that the mean Average Precision, is the average of all the average precision scores and subsequently evaluates the entire model across all object classes [35]. The mAP can be defined as [40]:

$$MAP = \frac{1}{n} \sum_{n} AP_n \tag{3.7}$$

3.3 Technical Specifications

The models were implemented on a DELL XPS 13-9365 machine that had a random access memory of 15.5 gigabytes, four intel i7-8500Y CPU of 1.5 GHz, a 64 bit windows 10 operating system [41], as well as a 512.1 gigabyte disk capacity. In addition to this the programming language that was used to implement these models was python [28], which used the python programming libraries of pandas [42],scikit learn [43], numpy [44], scipy [45], tensorflow [30], keras [29], matplotlib [46] amongst many others.

3.4 Limitations

Computational power is hindering in this regard as the implementation for the Single Stage Detectors and Two Stage Detectors takes a considerable amount of time to run especially, for large datasets. Furthermore it is prescribed that the models are implemented and executed with the help of a GPU. This is done as GPU's are able to make several times more computations than the CPU with regard to deep learning models. The Compute Unified Device Architecture (CUDA) toolkit [47], which was developed by the Nvidia corporation is the necessary API that employs the use of the GPU for deep learning models [47]. As such this toolkit can be used alongside other deep learning APIs such as Keras [29] and Pytorch [48]. In light of aforementioned CUDA toolkit and the prescribed GPU, it is necessary to note that the given GPU that has to be used in conjunction with the CUDA toolkit would be a Nvidia GPU. These GPUs have been historically used to build powerful gaming platforms, since these powerful systems need a dedicated electronic unit to quickly

render and create various images which are needed as an output for a display [47]. With this in mind the given system that will be used to undertake the given investigation does not have the necessary Nvidia GPU to work alongside the CUDA toolkit. This inevitably means that the CPU will be burdened with performing the necessary computations and executing the given models [47].

As with any research investigation, time constraints are also factored in. As already noted the CPU will be expected to carry out all the necessary computation for the proposed investigation and as such the run-times for the given models will be considerably longer/slower than what was generated in similar bodies of work.

Initially, it was planned that this investigation would seek to find out and understand how well each of the previously mentioned classifiers, performed when detecting vehicles/objects from satellite/remotely sensed images, across various datasets. Collecting this particular set of datasets, proved to be a difficult task, as a considerable amount of them were not annotated and labelled properly. This was the case for the aerial elephants dataset [49] as well as the Stanford drone dataset [50]. As such, given the time constraints of this project, it would be a time consuming process to label each image, thus changing the given datasets that were used, proved to be the better way to go.

3.5 Ethical Considerations

With regard to the ethics surrounding these datasets, there is not much to consider ethically as the datasets are publicly available and they do not infringe upon individual rights. No information is used to also personally identify individuals. The only cause for concern would be that this research area is typically used by surveillance states and institutions for unethical endeavours. With this in mind one has to weigh the possible ramifications (both positive and negative) about such research being conducted. The possible negatives of this is that authoritarian countries like China have been to known to impose draconian measures and maintain a police state with the aid of detection models [51]. This has already been demonstrated in places like Hong Kong where the surveillance systems were used to identify protesters which lead to a heavy handed use of force from the Hong Kong police in an effort to shut-down the Hong Kong protests. As such it follows that the research

conducted in this field will aid and supplement their agenda [52]. The positive takeaways from this research being conducted is that it contributes to the existing body of work which helps to advance machine learning algorithms and techniques related to object detection and image classification. As immense strides are being made in Artificial intelligence and robotics, technology related to self driving cars benefit immensely from these algorithms and models [15].

3.6 Conclusion

The basic structure, limitations and methodology for this investigation have been outlined and discussed. As such these steps and outlines need to be implemented in a timely fashion.

Chapter 4

Results, Discussion and Limitations

This chapter seeks to present and discuss the results of this investigation in relation to the four object detectors as well as the relevant datasets that were outlined. In addition to this, the limitations of the study will be presented in this chapter.

4.1 Results

The results of the four object detectors for all the datasets is presented below.

4.1.1 Results for the PASCAl VOC training dataset

The results of the four object detectors for the PASCAL VOC training dataset [17], are presented in the table 4.1 below, where the best performing model is highlighted in cyan.

TABLE 4.1: Table with the average precision(%) results for each object class and the subsequent mean average precision(%) score for each object detector, with regard to the PASCAL VOC training dataset [17].

	RCNN	Faster RCNN	YOLO	SSD
aeroplane	71.5	89.2	90.5	78.1
bicycle	68.1	82.4	81.4	84.8
bird	51.2	84.0	77.6	76.9
boat	39.7	58.3	62.6	69.0
bottle	30.0	58.8	53.7	47.1
bus	54.2	86.6	81.8	87.5
car	69.9	82.7	75.3	86.4
cat	29.3	88.2	86.5	89.7
chair	49.4	51.6	43.1	60.0
cow	45.2	87.1	85.7	88.8
dining-table	71.6	66.5	55.4	77.8
dog	57.0	92.9	81.8	87.2
horse	73.3	85.8	81.2	89.6
motorbike	63.1	87.0	85.0	84.3
person	75.7	87.2	79.6	78.7
potted-plant	55.1	49.5	48.8	51.0
sheep	27.3	75.9	83.4	78.3
sofa	38.7	68.1	64.9	82.8
train	60.2	89.4	83.7	90.0
tv-monitor	77.8	64.3	72.1	79.5
mAP	55.4	76.8	73.7	78.3

The results of the four object detectors for the PASCAL VOC testing dataset [17], are presented in the table 4.2 below, where the best performing model is highlighted in cyan.

TABLE 4.2: Table with the average precision(%) results for each object class and the subsequent mean average precision(%) score for each object detector, with regard to the PASCAL VOC testing dataset [17].

	RCNN	Faster RCNN	YOLO	SSD
aeroplane	68.3	88.6	86.0	77.5
bicycle	67.7	78.2	79.4	83.9
bird	45.8	80.7	75.8	75.2
boat	38.6	53.9	60.9	66.6
bottle	29.6	57.1	51.2	44.9
bus	51.0	82.6	79.5	85.5
car	69.4	79.4	71.6	84.7
cat	28.5	86.1	86.0	87.3
chair	48.1	49.5	42.8	58.4
cow	40.3	84.8	80.6	82.4
dining-table	68.4	65.9	54.2	76.1
dog	53.8	89.2	83.3	85.4
horse	71.3	84.3	79.7	86.7
motorbike	59.4	83.7	82.1	83.9
person	72.7	84.6	76.2	76.5
potted-plant	51.7	44.8	44.4	48.5
sheep	22.9	75.2	79.5	80.1
sofa	34.3	64.8	65.3	87.4
train	57.7	85.7	80.8	85.7
tv-monitor	75.8	59.1	68.7	76.3
mAP	52.8	73.9	71.1	76.1

The bar graph 4.1 below illustrates the mean average precision score (%) for each of the object detectors, for the PASCAL VOC dataset [17].

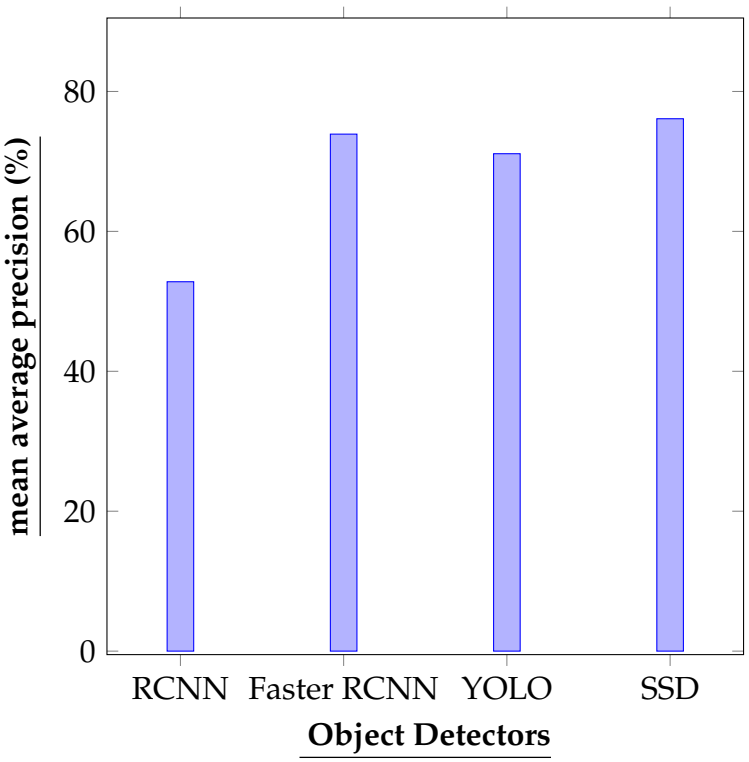


FIGURE 4.1: Bar Graph illustrating the mAP of each of the object detectors for the PASCAL VOC dataset [17].

From graph 4.1 and table 4.2, it can be seen that the SSD had the highest mAP score, followed by the Faster RCNN and YOLO, and as such the RCNN had the lowest mAP score.

4.1.2 Results for the Udacity self driving car training dataset

The results of the four object detectors for the Udacity self driving car training dataset [18], are presented in the table 4.3 below, where the best performing model is highlighted in cyan.

TABLE 4.3: Table with the average precision(%) results for each object class and the subsequent mean average precision(%) score for each object detector with regard to the Udacity self driving car training dataset [18].

	RCNN	Faster RCNN	YOLO	SSD
biker	27.0	13.8	35.3	38.9
car	44.1	67.5	72.0	55.7
traffic Light	38.5	32.1	44.7	51.3
Truck	55.6	59.0	62.8	63.5
pedestrian	31.0	59.5	63.2	56.9
mAP	39.2	46.4	55.7	53.1

The results of the four object detectors for the Udacity self driving car testing dataset [18], are presented in the table 4.4 below, where the best performing model is highlighted in cyan.

TABLE 4.4: Table with the average precision(%) results for each object class and the subsequent mean average precision(%) score for each object detector with regard to the Udacity self driving car testing dataset [18].

	RCNN	Faster RCNN	YOLO	SSD
biker	22.9	10.7	33.4	37.2
car	42.4	63.5	65.2	53.7
traffic Light	34.1	28.8	41.6	46.7
Truck	52.6	55.0	61.8	59.4
pedestrian	26.0	57.3	60.5	55.4
mAP	35.6	43.1	49.5	50.5

The bar graph 4.2 below illustrates the mean average precision score (%) for each of the object detectors, for Udacity self driving car testing dataset [18].

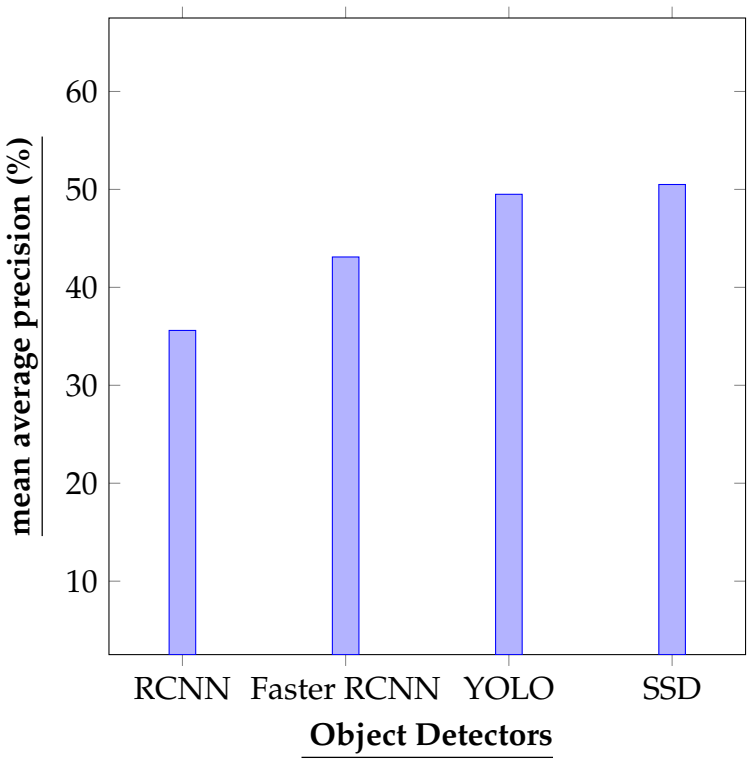


FIGURE 4.2: Bar Graph illustrating the mAP of each of the object detectors for the Udacity self driving car testing dataset [18].

From graph 4.2 and table 4.4, it can be seen that the SSD had the highest mAP score, followed by YOLO and the Faster RCNN, and as such the RCNN had the lowest mAP score.

4.1.3 Results for the Eastern North American 24 dataset

The results of the four object detectors for the Eastern North American 24 training dataset [19], are presented in the table 4.5 below, where the best performing model is highlighted in cyan.

TABLE 4.5: Table with the average precision(%) results for each object class and the subsequent mean average precision(%) score for each object detector with regard to the Eastern North American training dataset [19].

	RCNN	Faster RCNN	YOLO	SSD
American black bear	61.4	55.5	64.1	66.9
American crow	37.8	53.1	42.5	52.3
coyote	49.3	68.9	58.0	55.1
bobcat	69.0	74.5	59.7	74.2
Northern racoon	74.2	78.7	69.3	63.9
Eastern cottontail	27.1	44.3	58.0	76.1
red fox	24.6	42.9	33.4	74.9
Eastern Fox squirrel	39.7	56.2	51.7	77.2
Eastern chipmunk	37.2	50.0	74.1	69.7
Woodchuck	38.0	32.1	75.8	72.3
Virginia opossum	54.5	74.8	74.2	69.5
Striped skunk	44.9	47.4	63.1	37.0
White tailed deer	57.2	79.3	71.8	79.2
mAP	47.3	57.4	61.0	66.2

The results of the four object detectors for the Eastern North American 24 testing dataset [19], are presented in the table 4.6 below, where the best performing model is highlighted in cyan.

TABLE 4.6: Table with the average precision(%) results for each object class and the subsequent mean average precision(%) score for each object detector with regard to the Eastern North American testing dataset [19].

	RCNN	Faster RCNN	YOLO	SSD
American black bear	57.5	53.1	62.5	65.7
American crow	34.1	50.3	39.1	47.6
coyote	46.9	65.0	55.9	52.1
bobcat	62.5	70.9	56.0	69.8
Northern racoon	72.7	74.5	66.3	58.0
Eastern cottontail	24.0	40.8	53.2	73.4
red fox	20.2	40.4	30.8	69.7
Eastern Fox squirrel	36.7	53.0	47.7	72.3
Eastern chipmunk	34.4	48.9	71.4	66.3
Woodchuck	35.6	27.7	70.2	68.9
Virginia opossum	43.8	62.6	69.1	64.4
Striped skunk	39.0	46.1	60.0	31.0
White tailed deer	52.3	72.8	68.7	76.5
mAP	43.1	54.3	57.8	62.7

The bar graph 4.3 below illustrates the mean average precision score (%) for each of the object detectors, for the Eastern North American testing dataset [19].

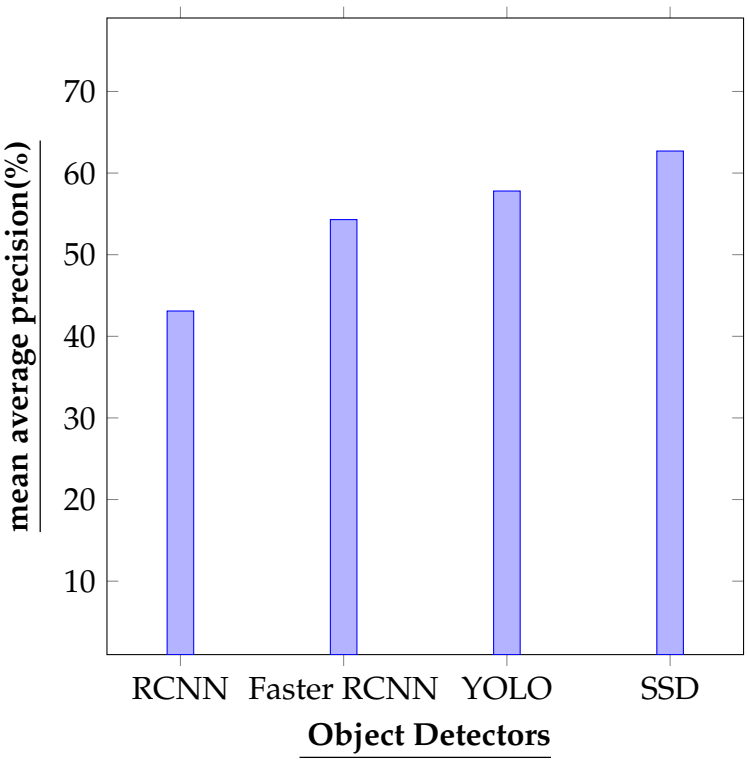


FIGURE 4.3: Bar Graph illustrating the mAP of each of the object detectors for the Eastern North American testing dataset [19].

From graph 4.3 and table 4.6, it can be seen that the SSD had the highest mAP score, followed by YOLO and the Faster RCNN , and as such the RCNN had the lowest mAP score.

4.1.4 Results for the Snapshot Serengeti training dataset

The results of the four object detectors for the Snapshot Serengeti training dataset [20], are presented in the table 4.7 below, where the best performing model is highlighted in cyan.

TABLE 4.7: Table with the average precision(%) results for each object class and the subsequent mean average precision(%) score for each object detector with regard to the snapshot Serengeti training dataset [20].

	RCNN	Faster RCNN	YOLO	SSD
buffalo	65.7	58.0	55.2	50..4
cheetah	24.1	27.3	42.7	33.9
eland	27.0	37.8	34.6	37.0
elephant	68.6	59.1	75.2	74.6
giraffe	43.2	65.4	27.1	39.2
hippopotamus	36.1	37.7	51.9	64.8
honeybadger	49.2	48.0	67.4	67.1
impala	41.8	58.1	78.3	65.4
Jackal	38.0	60.3	73.1	77.8
leopard	14.9	32.0	39.5	37.1
lion	58.1	66.8	54.0	59.5
rhinoceros	35.4	37.3	60.8	68.7
spotted hyena	59.3	38.7	48.1	51.4
topi	21.9	38.1	38.5	65.2
wildebeest	35.1	65.8	66.0	67.9
zebra	53.3	29.4	44.6	50.1
mAP	41.3	47.3	49.8	52.2

The results of the four object detectors for the Snapshot Serengeti testing dataset [20], are presented in the table 4.8 below, where the best performing model is highlighted in cyan.

TABLE 4.8: Table with the average precision(%) results for each object class and the subsequent mean average precision(%) score for each object detector with regard to the snapshot Serengeti testing dataset [20].

	RCNN	Faster RCNN	YOLO	SSD
buffalo	60.7	53.2	52.5	47.4
cheetah	22.2	23.5	33.1	31.9
eland	19.8	35.1	30.4	35.2
elephant	66.6	55.7	72.0	70.8
giraffe	39.4	60.0	24.8	30.1
hippopotamus	33.5	34.3	50.3	59.3
honeybadger	45.8	45.3	60.6	65.7
impala	34.7	55.7	70.2	61.0
Jackal	35.5	57.2	72.9	70.9
leopard	10.3	29.2	36.7	34.5
lion	56.8	63.4	47.0	49.1
rhinoceros	29.1	36.8	58.9	64.4
spotted hyena	57.0	36.6	47.4	48.6
topi	15.7	32.5	34.2	57.0
wildebeest	32.3	60.3	63.7	63.8
zebra	49.2	21.9	42.1	46.0
mAP	38.0	43.8	49.8	52.2

The bar graph 4.4 below illustrates the mean average precision score (%) for each of the object detectors, for the snapshot Serengeti testing dataset [20].

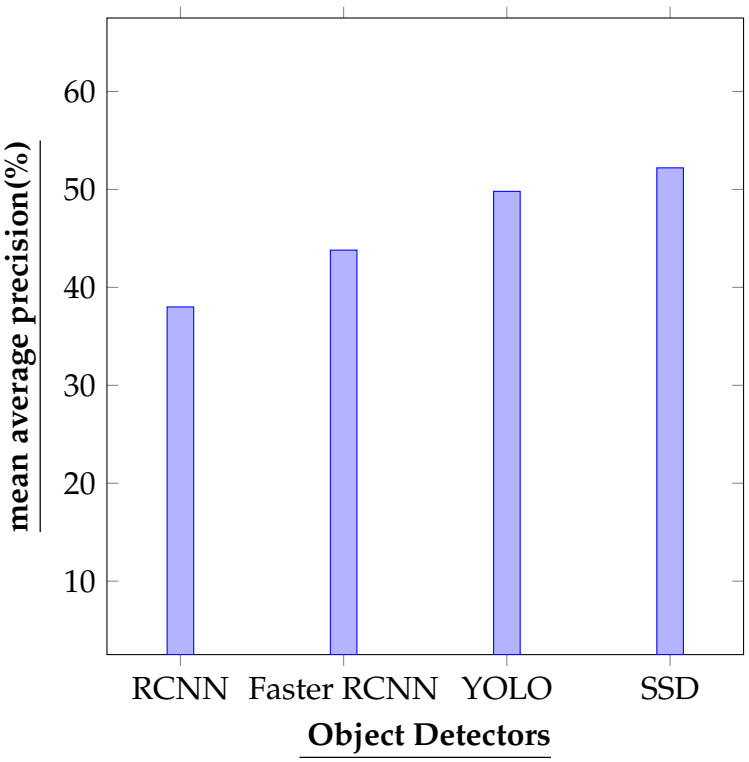


FIGURE 4.4: Bar Graph illustrating the mAP of each of the object detectors for the Serengeti Snapshot testing dataset [20].

From graph 4.4 and table 4.8, it can be seen that the SSD had the highest mAP score, followed by YOLO and the Faster RCNN , and as such the RCNN had the lowest mAP score.

4.1.5 Results for the DOTA dataset

The results of the four object detectors for the DOTA training dataset [21], are presented in the table 4.9 below, where the best performing model is highlighted in cyan.

TABLE 4.9: Table with the average precision(%) results for each object class and the subsequent mean average precision(%) score for each object detector with regard to the DOTA training dataset[21].

	RCNN	Faster RCNN	YOLO	SSD
baseball diamond	69.1	65.3	40.9	41.8
basketball court	57.0	57.9	56.2	31.5
bridge	47.2	34.6	27.9	81.4
ground track field	68.3	55.1	37.3	22.0
harbor	76.8	51.7	35.0	19.2
helicopter	79.0	34.8	14.4	7.4
large vehicle	94.3	47.5	41.8	39.2
plane	81.2	85.7	82.0	58.1
roundabout	55.1	46.1	37.6	37.4
ship	77.8	44.3	49.0	29.5
small vehicle	88.5	51.7	41.8	5.7
soccer ball field	75.8	40.4	47.4	12.9
storage tank	77.3	88.4	46.1	54.7
swimming pool	90.2	52.4	36.7	13.0
tennis court	87.1	74.9	60.5	86.3
mAP	75.0	55.4	43.6	36.0

The results of the four object detectors for the DOTA testing dataset [21], are presented in the table 4.10 below, where the best performing model is highlighted in cyan.

TABLE 4.10: Table with the average precision(%) results for each object class and the subsequent mean average precision(%) score for each object detector with regard to the DOTA testing dataset[21].

	RCNN	Faster RCNN	YOLO	SSD
baseball diamond	66.2	62.7	35.9	37.9
basketball court	53.3	54.0	51.5	27.1
bridge	43.4	31.6	26.8	77.7
ground track field	65.9	53.9	35.7	16.9
harbor	70.2	39.8	35.9	11.3
helicopter	67.0	29.6	12.0	0.9
large vehicle	86.2	44.4	39.0	35.1
plane	78.8	80.2	78.	56.4
roundabout	50.0	42.3	34.5	25.5
ship	73.3	41.9	46.8	25.3
small vehicle	84.2	48.3	36.1	0.0
soccer ball field	71.6	36.1	38.6	7.2
storage tank	74.6	73.9	43.7	51.6
swimming pool	87.8	47.4	32.6	8.4
tennis court	83.5	69.5	58.5	83.8
mAP	70.4	50.3	39.9	31.0

The bar graph 4.5 below illustrates the mean average precision score (%) for each of the object detectors, for the DOTA [21].

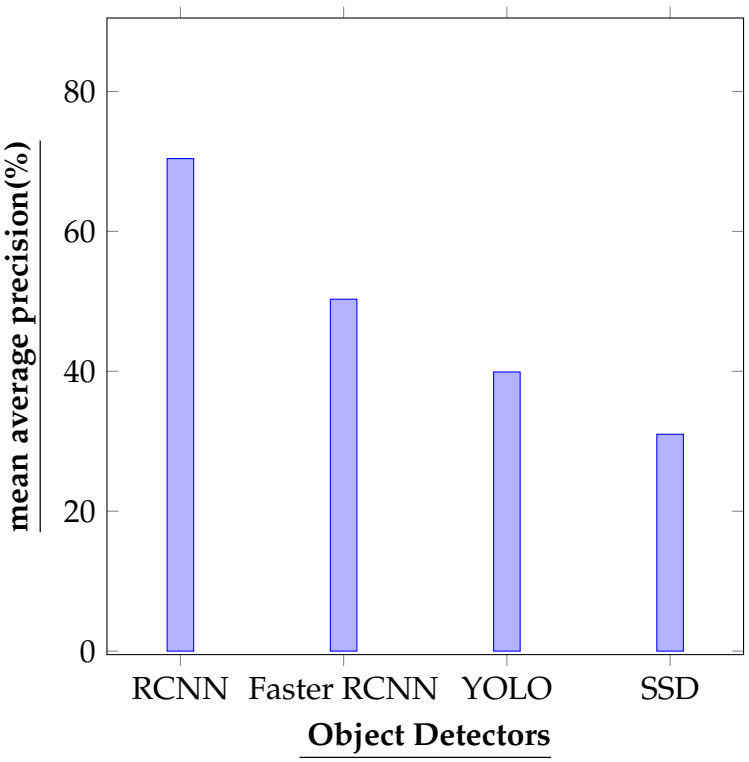


FIGURE 4.5: Bar Graph illustrating the mAP of each of the object detectors for the DOTA testing dataset [21].

From graph 4.5 and table 4.10, it can be seen that the RCNN had the highest mAP score, followed by the Faster RCNN and YOLO, and as such the SDD had the lowest mAP score.

4.1.6 Consolidation of results

From graph 4.6, the various performances of the object detectors can be seen across various the datasets that were used throughout this investigation.

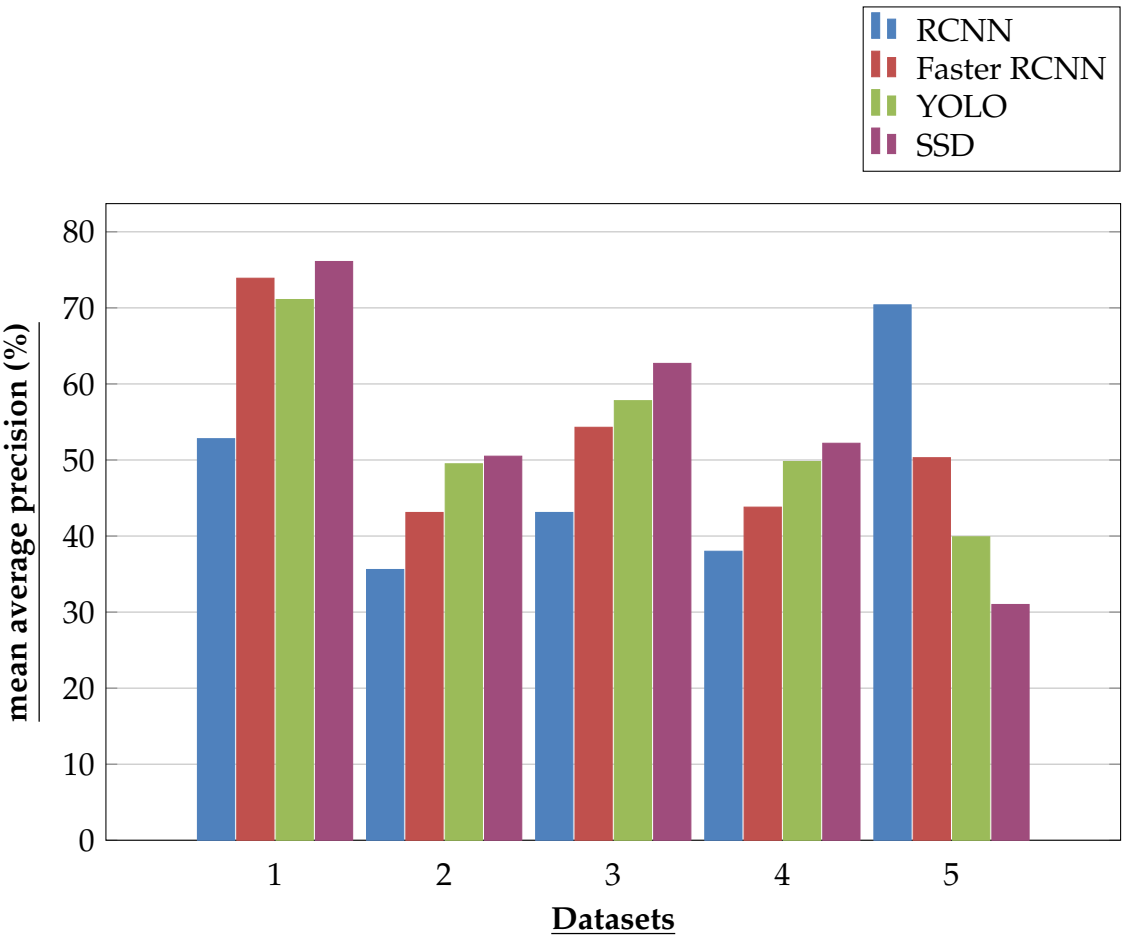


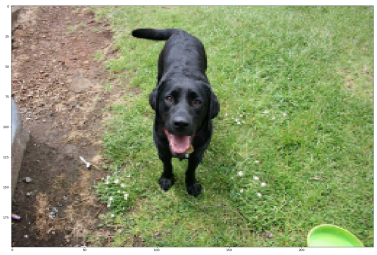
FIGURE 4.6: Bar graph denoting the object detectors across all the datasets. 1 denotes the Pascal VOC [17], 2 denotes the Udacity self driving car testing dataset [18], 3 denotes the Eastern Northern American 24 testing dataset [19], 4 denotes the Serengeti Snapshot testing dataset [20], and 5 denotes the DOTA testing dataset [21].

4.2 Discussion

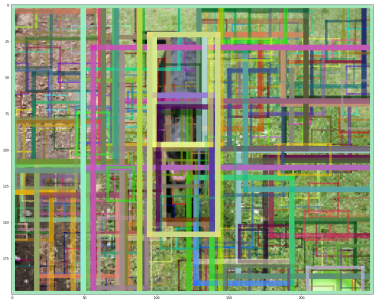
There are several factors and metrics that need to be outlined and discussed for each image classifier, in terms of how they performed across the aforementioned datasets. These will be outlined in the subsequent sections that follow as they relate to the overall performance, accuracy and general functionality of each of the classifier.

4.2.1 Regions with Convolutional Neural Networks

Upon implementing the RCNN, it is noted that the network generates candidate regions for each image through the use of a selective search algorithm. The number of regions generated varies from image to image but generally speaking approximately 2000 candidate regions are produced for each image. Below one can see an example of the candidate regions being generated from each dataset.



(A) An image before candidate regions are generated.

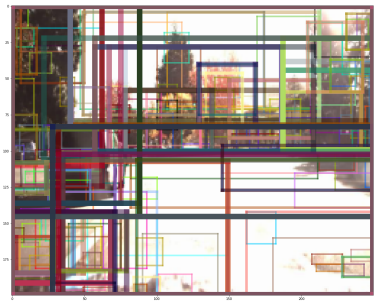


(B) An image after candidate regions are generated.

FIGURE 4.7: An example of RCNN generating candidate regions on an image from PASCAL VOC [17].



(A) An image before candidate regions are generated.

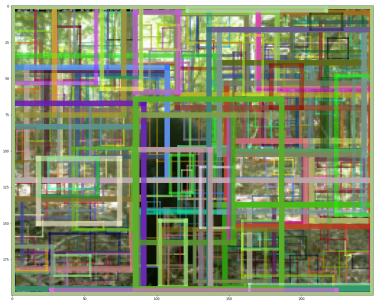


(B) An image after candidate regions are generated.

FIGURE 4.8: An example of RCNN generating candidate regions on an image from the Udacity self driving car dataset [18].



(A) An image before candidate regions are generated.

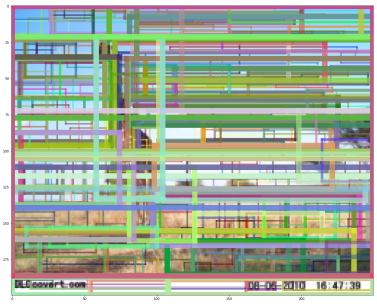


(B) An image after candidate regions are generated.

FIGURE 4.9: An example of RCNN generating candidate regions on an image from the Eastern North American dataset [19].

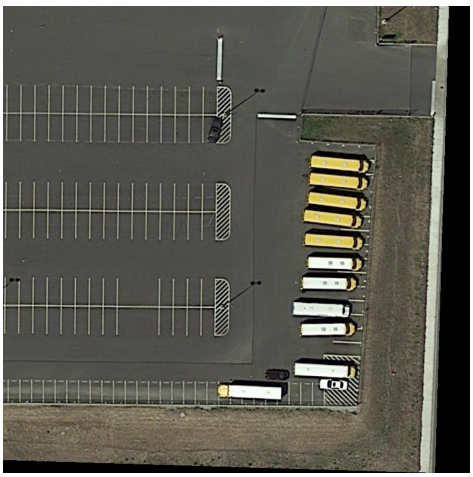


(A) An image before candidate regions are generated.

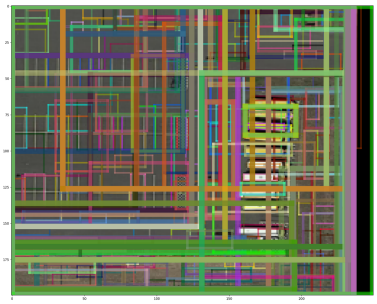


(B) An image after candidate regions are generated.

FIGURE 4.10: An example of RCNN generating candidate regions on an image from the snapshot Serengeti dataset [20].



(A) An image before candidate regions are generated.



(B) An image after candidate regions are generated.

FIGURE 4.11: An example of RCNN generating candidate regions on an image from the DOTA dataset [21].

In order to make the necessary predictions an IoU threshold is applied and in this case, an IoU threshold of 0.4 was used. With this in mind the overall training time was considerably long given that the predictions were being made through the sole use of a CPU, which further added to the time and computational complexity. As such the average training time for the RCNN model with regard to the datasets

used, was approximately 268 hours. To further elucidate the average runtime at test time was approximately 440 seconds per image. That is it took an average of 440 seconds to generate the necessary candidate regions and subsequently make the relevant prediction.

Moving onto the aspect of accuracy, what is of particular interest in this regard would be the mAP that was achieved for each dataset. The RCNN had the highest mAP in regards to the DOTA dataset [21], followed by the PASCAL VOC [17], the Eastern North American dataset [19], the snapshot Serengeti dataset [20], and lastly the Udacity self driving car [18]. The differences in performance across datasets can be attributed to costs related to the feature maps in relation to storage space as well as how localizations are made for small/large objects within the image. To elucidate further a significant drawback of the RCNN is that the features from each proposal from each image is written to a storage device and as such these features require a large volume of storage space. This drawback significantly impacts training time. Furthermore it is observed that the generated regions (through the use of the selective search algorithm) were able to better detect small objects within images (as observed with the DOTA dataset [21]).

The RCNN model implemented in this paper, outperformed the RCNN implemented in Girshick et al. [5], in relation to the PASCAL VOC dataset [17]. The mAP achieved for the RCNN model implemented in [5], for the PASCAL VOC dataset, was 50.2%, whereas the mAP achieved by the RCNN model implemented in this paper was 52.8%. In relation to the other datasets, it does not seem that the RCNN was applied to them in prior works.

4.2.2 Faster Regions with Convolutional Neural Networks

With regards to the Faster RCNN, object proposals are delegated to the RPN and as such approximately 300 regions were being generated for each image when making the necessary predictions. IoU thresholds of 0.7 as well as 0.3 were considered in order to adequately assign the appropriate negative/positive anchor boxes. The proposed regions were used to train the Fast RCNN.

As noted before this unified system leads to quicker training and testing times. In regards to training time, it is noted that this model had an average training time of approximately 178 hours across the four datasets and an average runtime of approximately 20 seconds (in relation to the test images).

The Faster RCNN recorded its highest mAP with the PASCAL VOC dataset [17], followed by the DOTA dataset [21], the Eastern North American dataset [19], the snapshot Serengeti dataset [20], and lastly the Udacity self driving car [18]. Once again it is observed that implementations that use regional proposals are better able to detect small objects within images due to the results obtained from the DOTA dataset [21].

In order to improve and account for the performance of the Faster RCNN there are perhaps several steps or adjustments in the given methodology one could make. One of the first options would be to address the issue of training networks with features shared. For this investigation, an alternate training methodology was used and as such experimenting with an approximate joint training implementation as well as a non approximate joint training implementation could improve the overall performance of the given model (more so in terms of training time) [11]. Moving on, a change in the settings of the anchors could be used to improve the results. This is done by changing the number of aspect ratios and anchor scales as well as their given values. The use of different anchor scales and anchors can be done since the scales and aspect ratios are disentangled dimensions for the detection accuracy. Lastly one could also consider that the λ in 2.1 could be changed to different values in order to experiment and moderately improve results but in this regard the λ is considered to be insensitive within a wide range. This is due to the fact that within a scale of two orders of magnitude, λ only illustrates marginal increases in performance [11].

The Faster RCNN model implemented in this paper, outperformed the Faster RCNN model implemented in Ren et al. [11], in relation to the PASCAL VOC dataset [17]. The mAP achieved for the Faster RCNN model implemented in [11], for the PASCAL VOC dataset [17], was 73.2%, whereas the mAP achieved by the Faster RCNN model implemented in this paper was 73.9%.

In Schneider, Taylor, and Kremer [53], the Faster RCNN model implemented in that paper achieved an mAP of 76.7%, whereas the Faster RCNN model implemented in this paper achieved an mAP of 43.8%, in relation to the Snapshot Serengeti dataset [20].

In Xia et al. [21], an mAP of 60.46% (using oriented bounding boxes) and 54.13% (using horizontal bounding boxes), in relation to the DOTA dataset [21], was achieved by using a SSD model [21]. The model implemented in this paper achieved a mAP of 50.3%, and as such was outperformed by the SSD models implemented in Xia et al. [21], that used two variations for their bounding boxes (horizontal and oriented bounding boxes) [21]. In relation to the other datasets, it does not seem that the Faster RCNN was applied to them, in prior works.

4.2.3 You Only Look Once

As a single stage detector YOLO uses the full image when training and when it makes its' subsequent predictions. The first thing to consider with this implemented version of YOLO is that K clustering was used to generate the necessary anchor boxes. For each of the datasets the following hyper-parameter of $k \in [2, 10]$ was used. In order to determine the number clusters, the elbow curve is used to determine this. In regards to the training and testing time, YOLO had an average training time of approximately 72 hours across all 5 datasets and had an average test time of approximately 9 seconds to generate the necessary predictions on a given image. It is worth noting that the IoU threshold that was used in this investigation was an IoU of 0.5. Again, it would have been more appropriate to adjust this parameter in regards to each relevant dataset.

YOLO had its' highest mAP with the PASCAL VOC [17], followed by the Eastern North American dataset [19], the snapshot Serengeti dataset [20], the Udacity self driving car [18], and lastly the DOTA dataset [21]. There were several steps that could have been implemented in order to improve the overall performance of the

YOLO model in regards to the other datasets that it achieved poor results on. Firstly a pass-through layer could have been added so as to improve the localization of objects with small bounding boxes (that is images with small objects). To further improve the performance of this given YOLO implementation, the model could be trained using images at a higher resolution of 448×448 instead of 224×224. Such a change will prevent the model from switching to object detection while also simultaneously adjusting to the current input resolution [1]. Lastly multi-scale training could be employed in order to train the model using different images at different input scales which would allow the network to better perform across different input dimensions [1].

The YOLO model implemented in this paper, outperformed the YOLO model implemented in Redmon et al. [12], but was outperformed by the YOLOv2 model implemented in Redmon and Farhadi [1], in relation to the PASCAL VOC dataset [17]. The mAP achieved for [12] for the PASCAL VOC dataset, was 57.9%, whereas the mAP achieved by the YOLO model implemented in this paper was 71.1%, while the mAP achieved by the YOLOv2 implemented in Redmon and Farhadi [1] was 73.4%.

In Schneider, Taylor, and Kremer [53], the YOLO model implemented in that paper achieved a mAP of 43.3% whereas the YOLO model implemented in this paper achieved an mAP of 49.8%, in relation to the Snapshot Serengeti dataset [20].

In Xia et al. [21], an mAP of 25.492% (using oriented bounding boxes) and 39.2% (using horizontal bounding boxes), in relation to the DOTA dataset [21], was achieved by using a YOLOv2 model [21]. The model implemented in this paper achieved a mAP of 39.9%, which outperformed the YOLOv2 model that was implemented in Xia et al. [21], that used two variations for their bounding boxes (horizontal and oriented bounding boxes) [21]. In relation to the other datasets, it does not seem that the YOLO was applied to them in prior works.

4.2.4 Single Shot MultiBox Detector

When the SSD was implemented it managed an average training time of approximately 68 hours across all four datasets, while it also had an average test time of approximately 8 seconds to generate the necessary predictions on a given image. The IoU threshold that was used with regard to the SSD was an IoU of 0.7. Further adjustments or expansions on this particular aspect could be made in subsequent bodies of work.

The SSD had its' highest mAP with the PASCAL VOC [17], followed by the Eastern North American dataset [19],the snapshot Serengeti dataset [20], the Udacity self driving car [18], and lastly the DOTA dataset [21]. The SSD model is cited as being prone to poorer performances where objects have a small bounding box [13]. As such with this in mind the Udacity [18], Serengeti snapshot [20], the Eastern North American [19] ands the DOTA [21] datasets highlight this in regards to the SSD's poorer performance on these datasets. With this in mind SSD will conversely perform better with large objects (like those in PASCAL VOC [17])[13].

In order to generate the required results, the SSD was implemented in such a manner that i)t only receives images, with a 300×300 resolution, as an input. Taking this into account using a higher resolution input might have improved the models performance but impacted its' computational and time complexities in a negative manner. With this in mind there are several other data augmentation strategies that could be implemented in order to improve the results across the four datasets. The first strategy one could use would be to, place an image on a canvas that is 16 times larger than the original image and subsequently filled with mean values. This expansion of the images would introduce more training examples for the model but it would subsequently increase training time as the number of iterations would increase. This strategy is implemented so as to improve the accuracy related to small objects [13]. Another strategy that could be employed would be to improve the tiling of default boxes. This would be done so that the position and scale of default boxes would have a better alignment with their receptive fields for every region on a given feature map.

When comparing the results obtained in this investigation, in relation to the PASCAL VOC dataset [17], to the results obtained in Liu et al. [13], it needs to be noted that the model implemented by Liu et al. [13], had two variations [13]. The first implementation had an input image of 300×300 and the second implementation had an input image of 512×512 [13]. The first SSD implementation (which had an mAP of 74.3%), was outperformed by the SSD model implemented in this paper (which had an mAP of 76.1%), however in light of this, the second SSD implementation from Liu et al. [13] had an mAP of 76.8%, which marginally outperformed the SSD model implemented in this paper.

In Xia et al. [21], an mAP of 29.86% (using oriented bounding boxes) and 17.84% (using horizontal bounding boxes), in relation to the DOTA dataset [21], was achieved by using a SSD model [21]. The model implemented in this paper achieved a mAP of 31.0%, which outperformed the SSD model that was implemented in Xia et al. [21], that used two variations for their bounding boxes (horizontal and oriented bounding boxes) [21]. In relation to the other datasets, it does not seem that the SSD was applied to them in prior works.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This investigation sought to compare and contrast the results/performance that single and two stage detectors would obtain across several datasets. In light of this it is observed that single stage detectors generally outperformed and generated better results than two stage detectors. The notable exception to this would be although two stage detectors are still quite inefficient in making timely predictions, they still had a better mAP score when making predictions on remotely sensed images. In this regard, the RCNN followed by the Faster RCNN (Two stage detectors) vastly outperformed YOLO and the SSD (Single stage detectors). When considering the performance of these implementations, on image datasets that were not remotely sensed, one can actually see a reverse in performance as the single stage detectors (SSD followed by YOLO) outperform the two stage detectors (Faster RCNN followed RCNN).

The differences in performance with regard to remotely and none remotely sensed image datasets, are largely due to the ability to perform localizations on small objects within the given images. The RCNN and Faster RCNN utilize regional proposals in order to generate their predictions and in light of this they are better able to localize small objects within images.

5.2 Future Work

This body of work could be further extended and expanded upon by including contemporary methodologies and implementations such as the mask-RCNN [37], YOLOv3 [54], YOLOv4 [55]. Furthermore the use of different remotely sensed datasets could be considered in order to have more expansive results. Furthermore, the limitations of the study could also be accounted in order to generate more meaningful results, more so in relation to the optimization of the hyper-parameters.

Another consideration that could be taken into account would be to improve the overall performance of two stage detectors, in respect to the amount of time needed for them to make predictions as well as improving the overall accuracy of single stage detectors on remotely sensed images.

One main limitation of this study would be that the hyper-parameters for the models used in this investigation were not optimized, for the various datasets that were used in this investigation. To alleviate this limitation, several algorithms can be implemented in order to fine-tune and optimise the hyper-parameters of the object detector models that were implemented [56]. Search algorithms such as Grid search, Random search and the Bayesian optimization are just some of the many search algorithms that can be considered for future bodies of work [56]. In addition to this, open and closed source hyper-parameter toolkits can also be employed and used in order to streamline the fine tuning of the hyper-parameters in question [56].

Bibliography

[1] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. arXiv: 1612.08242 [cs.CV].

[2] Constantine Papageorgiou and Tomaso Poggio. "A Trainable System for Object Detection". In: *International Journal of Computer Vision* 38 (June 2000), pp. 15–33. DOI: 10.1023/A:1008162616689.

[3] Henry Schneiderman and Takeo Kanade. "Object Detection Using the Statistics of Parts". In: *International Journal of Computer Vision* 56 (Feb. 2004), pp. 151–177. DOI: 10.1023/B:VISI.0000011202.85607.00.

[4] N. Dalal and B. Triggs. "Histograms of oriented gradients for human detection". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.

[5] Ross B. Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *CoRR* abs/1311.2524 (2013). arXiv: 1311.2524. URL: http://arxiv.org/abs/1311.2524.

[6] Kunihiko Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological cybernetics* 36 (Feb. 1980), pp. 193–202. DOI: 10.1007/BF00344251.

[7] Yann Lecun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1 (Dec. 1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, 1097–1105.

[9] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. "Deep Neural Networks for Object Detection". In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'13. Lake Tahoe, Nevada: Curran Associates Inc., 2013, 2553–2561.

[10] Ross Girshick. *Fast R-CNN*. 2015. arXiv: 1504.08083 [cs.CV].

[11] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2015. arXiv: 1506.01497 [cs.CV].

[12] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2015. arXiv: 1506.02640 [cs.CV].

[13] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *Lecture Notes in Computer Science* (2016), 21–37. ISSN: 1611-3349. DOI: 10.1007/978-3-319-46448-0_2. URL: http://dx.doi.org/10.1007/978-3-319-46448-0_2.

[14] Changqing Cao et al. "Research on Airplane and Ship Detection of Aerial Remote Sensing Images Based on Convolutional Neural Network". In: *Sensors* 20 (Aug. 2020), p. 4696. DOI: 10.3390/s20174696.

[15] Yoganandhan .A et al. "Fundamentals and Development of Self-Driving Cars". In: *Materials today: proceedings* (May 2020). DOI: 10.1016/j.matpr.2020.04.736.

[16] Chinthakindi Balaram Murthy et al. "Investigations of Object Detection in Images/Videos Using Various Deep Learning Techniques and Embedded Platforms—A Comprehensive Review". In: *Applied Sciences* 10.9 (2020). ISSN: 2076-3417. DOI: 10.3390/app10093280. URL: https://www.mdpi.com/2076-3417/10/9/3280.

[17] Roozbeh Mottaghi et al. "The Role of Context for Object Detection and Semantic Segmentation in the Wild". In: *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*. CVPR '14. USA: IEEE Computer Society, 2014, 891–898. ISBN: 9781479951185. DOI: 10.1109/CVPR.2014.119. URL: https://doi.org/10.1109/CVPR.2014.119.

[18] Udacity. "Udacity Self-Driving Car Driving Data". In: (2016). URL: https://github.com/udacity/self-driving-car.

[19] Hayder Yousif, Roland Kays, and Zhihai He. "Dynamic Programming Selection of Object Proposals for Sequence-Level Animal Species Classification in the Wild". In: *IEEE Transactions on Circuits and Systems for Video Technology* (2019).

[20] AB Swanson et al. *Data from: Snapshot Serengeti, high-frequency annotated camera trap images of 40 mammalian species in an African savanna*. 2015. DOI: doi:10.5061/dryad.5pt92. URL: https://doi.org/10.5061/dryad.5pt92.

[21] Gui-Song Xia et al. *DOTA: A Large-scale Dataset for Object Detection in Aerial Images*. 2017. arXiv: 1711.10398 [cs.CV].

[22] Yangqing Jia et al. *Caffe: Convolutional Architecture for Fast Feature Embedding*. 2014. arXiv: 1408.5093 [cs.CV].

[23] J. R. R. Uijlings et al. "Selective Search for Object Recognition". In: *International Journal of Computer Vision* 104.2 (2013), pp. 154–171. URL: https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013.

[24] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *Int. J. Comput. Vision* 115.3 (Dec. 2015), 211–252. ISSN: 0920-5691. DOI: 10.1007/s11263-015-0816-y. URL: https://doi.org/10.1007/s11263-015-0816-y.

[25] Alan Hevner and Samir Chatterjee. "Design Science Research in Information Systems". In: *Design Research in Information Systems: Theory and Practice*. Vol. 22. Boston, MA: Springer US, 2010, pp. 9–22. ISBN: 978-1-4419-5653-8. DOI: 10.1007/978-1-4419-5653-8_2. URL: https://doi.org/10.1007/978-1-4419-5653-8_2.

[26] Alan R. Hevner. "A Three Cycle View of Design Science Research". In: *Scandinavian Journal of Information Systems* 19 (2007), p. 4.

[27] Yaneer Bar-Yam. "The sciences of the artificial, 3rd edition: By Herbert A. Simon". In: *Complexity* 3.5 (1998), pp. 47–48. DOI: https://doi.org/10.1002/(SICI)1099-0526(199805/06)3:5<47::AID-CPLX7>3.0.CO;2-F. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/%28SICI%291099-0526%28199805/06%293%3A5%3C47%3A%3AAID-CPLX7%3E3.0.CO%3B2-F. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291099-0526%28199805/06%293%3A5%3C47%3A%3AAID-CPLX7%3E3.0.CO%3B2-F.

[28] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[29] François Chollet et al. *Keras*. https://keras.io. 2015.

[30] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[31] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. arXiv: 1409.1556 [cs.CV].

[32] J. Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshops)*. Los Alamitos, CA, USA: IEEE Computer Society, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848. URL: https://doi.ieeecomputersociety.org/10.1109/CVPR.2009.5206848.

[33] Robert Tibshirani, Guenther Walther, and Trevor Hastie. "Estimating the number of clusters in a data set via the gap statistic". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63.2 (2001), pp. 411–423. DOI: 10.1111/1467-9868.00293. eprint: https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/1467-9868.00293. URL: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/1467-9868.00293.

[34] Pan Zhou et al. "Towards Theoretically Understanding Why SGD Generalizes Better Than ADAM in Deep Learning". In: *CoRR* abs/2010.05627 (2020). arXiv: 2010.05627. URL: https://arxiv.org/abs/2010.05627.

[35] Paul Henderson and Vittorio Ferrari. "End-to-end training of object class detectors for mean average precision". In: *CoRR* abs/1607.03476 (2016). arXiv: 1607.03476. URL: http://arxiv.org/abs/1607.03476.

[36] Ethan Zhang and Yi Zhang. "Average Precision". In: *Encyclopedia of Database Systems*. Ed. by LING LIU and M. TAMER ÖZSU. Boston, MA: Springer US, 2009, pp. 192–193. ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_482. URL: https://doi.org/10.1007/978-0-387-39940-9_482.

[37] Kaiming He et al. "Mask R-CNN". In: *CoRR* abs/1703.06870 (2017). arXiv: 1703.06870. URL: http://arxiv.org/abs/1703.06870.

[38] Hamid Rezatofighi et al. “Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 658–666. DOI: 10.1109/CVPR.2019.00075.

[39] David Olson and Dursun Delen. *Advanced Data Mining Techniques*. Jan. 2008. ISBN: 978-3-540-76916-3. DOI: 10.1007/978-3-540-76917-0.

[40] Steven M. Beitzel, Eric C. Jensen, and Ophir Frieder. “MAP”. In: *Encyclopedia of Database Systems*. Ed. by LING LIU and M. TAMER ÖZSU. Boston, MA: Springer US, 2009, pp. 1691–1692. ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_492. URL: https://doi.org/10.1007/978-0-387-39940-9_492.

[41] Ed Bott and Craig Stinson. *Windows 10 inside out*. Microsoft Press, 2019.

[42] Wes McKinney et al. “Data structures for statistical computing in python”. In: *Proceedings of the 9th Python in Science Conference*. Vol. 445. Austin, TX. 2010, pp. 51–56.

[43] Fabian Pedregosa et al. “Scikit-learn: Machine learning in Python”. In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[44] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585 (2020), 357–362. DOI: 10.1038/s41586-020-2649-2.

[45] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[46] John D Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in science & engineering* 9.3 (2007), pp. 90–95.

[47] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. *CUDA, release: 10.2.89*. 2020. URL: https://developer.nvidia.com/cuda-toolkit.

[48] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[49] Johannes J. Naudé and Deon Joubert. *The Aerial Elephant Dataset*. May 2019. DOI: 10.5281/zenodo.3234780. URL: https://doi.org/10.5281/zenodo.3234780.

[50] Alexandre Robicquet et al. "Learning Social Etiquette: Human Trajectory Understanding In Crowded Scenes". In: vol. 9912. Oct. 2016, pp. 549–565. ISBN: 978-3-319-46483-1. DOI: 10.1007/978-3-319-46484-8_33.

[51] S. Feldstein and Carnegie Endowment for International Peace. *The Global Expansion of AI Surveillance*. Carnegie Endowment for International Peace, 2019. URL: https://books.google.be/books?id=W9JQzQEACAAJ.

[52] Stuart Hargreaves. "Online Monitoring of 'Localists' in Hong Kong: A Return to Political Policing?" In: *Surveillance & Society* 15 (Aug. 2017), p. 425. DOI: 10.24908/ss.v15i3/4.6619.

[53] Stefan Schneider, Graham W. Taylor, and Stefan C. Kremer. "Deep Learning Object Detection Methods for Ecological Camera Trap Data". In: *CoRR* abs/1803.10842 (2018). arXiv: 1803.10842. URL: http://arxiv.org/abs/1803.10842.

[54] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". In: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

[55] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 [cs.CV].

[56] Tong Yu and Hong Zhu. *Hyper-Parameter Optimization: A Review of Algorithms and Applications*. 2020. arXiv: 2003.05689 [cs.LG].