Universidade do Minho
Escola de Engenharia
Departmento de Informática

João Simões Farinha

In-Vehicle Object Detection with YOLO Algorithm

November 2018

Universidade do Minho
Escola de Engenharia
Departmento de Informática

João Simões Farinha

In-Vehicle Object Detection with YOLO Algorithm

Master Thesis
Computer Science Engineering

Thesis supervised by:
Paulo Cortez
André Ferreira

November 2018

ACKNOWLEDGEMENTS

I would like to thank my mentors Paulo Cortez and André Ferreira for their guidance during the writing of this thesis. I would also like to thank my family for their continued support throughout the development of this thesis.

ABSTRACT

With the growing computational power that we have at our disposal and the ever-increasing amount of data available the field of *machine learning* has given rise to *deep learning*, a subset of machine learning algorithms that have shown extraordinary results in a variety of applications from natural language processing to computer vision. In the field of computer vision, these algorithms have greatly improved the state-of-the-art accuracy in tasks associated with object recognition such as detection. This thesis makes use of one of these algorithms, specifically the YOLO algorithm, as a basis in the development of a system capable of detecting objects laying inside a car cockpit. To this end a dataset is collected for the purpose of training the YOLO algorithm on this task.

A comparative analysis of the detection performance of the YOLOv2 and YOLOv3 architectures is performed.Several experiments are performed by modifying the YOLOv3 architecture to attempt to improve its accuracy. Specifically tests are performed in regards to network size, and the multiple outputs present in this network. Explorative experiments are done in order to test the effect that parallel network might have on detection performance. Lastly tests are done to try to find an optimal learning rate and batch size for our dataset on the new architectures.

RESUMO

Com o crescente poder computacional que temos à nossa disposição e o aumento da quantidade dados a que temos acesso o campo de *machine learning* deu origem ao *deep learning* um subconjunto de algoritmos de *machine learning* que têm demonstrado resultados extraordinários numa variedade de aplicações desde processamento de linguagens naturais a visão por computador. No campo de visão por computador estes algoritmos têm levado a enormes progressos na correção de sistemas de deteção de objetos. Nesta tese usamos um destes algoritmos, especificament o YOLO, como base para desenvolver um sistema capaz de detetar objetos dentro de um carro. Dado isto um dataset é recolhido com o proposito de treinar o algoritmo YOLO nesta tarefa.

Uma analise comparativa da correção dos algoritmos YOLOv2 e YOLOv3 é realizada. Várias técnicas relacionadas com a modificação da arquitetura YOLOv3 são exploradas para otimizar o sistema para o problema especifico de deteção a bordo de veiculos. Especificamente testes são realizados no contexto de tamanho da rede e dos multiplos outputs presentes nesta rede. Experiencias exploratorias são realizadas de forma a testar o efeito que redes parallelas podem ter na correção dos algoritmos. Por fim testes são feitos para tentar encontrar learning rates e batch sizes apropriados para o nosso dataset nas novas arquiteturas.

CONTENTS

1 INTRODUCTION 2
- 1.1 Goals 2
- 1.2 Document Outline 3

2 STATE OF THE ART 4
- 2.1 Bibliographic Search Strategy 4
- 2.2 Computer Vision 4
- 2.3 Machine Learning 5
- 2.4 Deep Learning 7
- 2.5 Convolutional Neural Networks 8
- 2.6 Relevant Literature 13

3 PROBLEM AND ITS CHALLENGES 15
- 3.1 Datasets 16
- 3.2 Data Augmentation 19
- 3.3 The mAP Metric 19
- 3.4 The Darknet Framework and the YOLO Algorithm 20
- 3.5 YOLOv3 Network Architecture 21
- 3.6 Training Stopping Point 24
- 3.7 Environment Description 24

4 EXPERIMENTAL RESULTS 25
- 4.1 Comparison between YOLOv2 and YOLOv3 25
- 4.2 Removal of Layers 27
- 4.3 Smaller Size Network 28
- 4.4 Feature Pyramids and Multiple Paths 34
- 4.5 Parallel Networks 47
- 4.6 Learning Rate and Batch Size 54

5 CONCLUSIONS AND FUTURE WORK 57

References 59

LIST OF FIGURES

Figure 1 Example of the output of an object detection algorithm with bounding boxes around each object present in image with respective classification and the confidence of the algorithm on the result, adapted from (Redmon & Farhadi, 2018) 5
Figure 2 Artificial Neural Network with 2 hidden layers 8
Figure 3 Inception Module, adapted from (Szegedy et al., 2015) 10
Figure 4 Visualizing Learned features in a CNN and examples of images that maximize neuron activation in different layers, adapted from (Zeiler & Fergus, 2014) 12
Figure 5 Example picture from the IVOD dataset 18
Figure 6 Examples of augmented images 19
Figure 7 Graphical observation of the learning process during training for architectures with missing layers 28
Figure 8 Graphical observation of the learning process during training for architectures with smaller networks based on YOLOv3 32
Figure 9 Graphical observation of the learning process during training with smaller networks based on YOLOv3 using smallCOCO 33
Figure 10 Graphical observation of the learning process during training with smaller networks based on YOLOv3 using COCO 34
Figure 11 Graphical observation of the learning process during training with miniyolo without multiple paths 38
Figure 12 Graphical observation of the learning process during training with miniyolo without multiple paths with smallCOCO dataset 39
Figure 13 Graphical observation of the learning process during training with miniyolo with some paths removed 43
Figure 14 Graphical observation of the learning process during training with miniyolo with extra paths added 46
Figure 15 Graphical observation of the learning process during training with new architectures developed without shortcut layers 50
Figure 16 Graphical observation of the learning process during training with parallel networks 54
Figure 17 Graphical observation of the learning process during training for learning rate and batch size testing 55

LIST OF TABLES

Table 1	Relevant Literature	13
Table 2	The classes of the IVOD dataset	17
Table 3	The classes of the smallCOCO dataset	18
Table 4	YOLOv3 detection performance compared to other Neural Networks, adapted from (Redmon & Farhadi, 2018)	22
Table 5	The YOLOv3 architecture	23
Table 6	Training machine specifications	24
Table 7	The YOLOv2 architecture	26
Table 8	YOLOv2 and YOLOv3 comparison, number of iterations between parenthesis	27
Table 9	Individual results for testing with missing layers	28
Table 10	The miniyolo1 architecture	30
Table 11	The miniyolo2 architecture	31
Table 12	Individual results for testing with smaller networks based on YOLOv3	32
Table 13	Individual results for testing with smaller networks based on YOLOv3 with small-COCO	33
Table 14	Individual results for testing with smaller networks based on YOLOv3 with COCO	34
Table 15	The miniyolo1-1path architecture	36
Table 16	The miniyolo1-nopath architecture	37
Table 17	Individual results for tests with miniYOLO without multiple paths	38
Table 18	Individual results for miniyolo without multiple paths tests on the smallCOCO dataset	39
Table 19	The miniyolo1-12path architecture	41
Table 20	The miniyolo1-13path architecture	42
Table 21	Individual results for tests with miniyolo with some paths removed	43
Table 22	The miniyolo2-4paths architecture	44
Table 23	The miniyolo2-5paths architecture	45
Table 24	The miniyolo2-5paths architecture (continuation)	46
Table 25	Individual results tests with miniyolo with extra paths added	46
Table 26	The newDarkNetwork architecture	48
Table 27	The miniDarkNetwork architecture	49
Table 28	Individual results for tests with new architectures developed without shortcut layers	50
Table 29	The parallelMiniDark architecture	52
Table 30	The parallelMiniDarkFusion architecture	53

Table 31 Individual results for parallel networks tests 54
Table 32 Individual results for learning rate and batch size tests 56

LIST OF ACRONYMS

CNN	Convolutional Neural Network
ANN	Artificial Neural Network
DNN	Deep Neural Network
mAP	mean Average Precision
IoU	Intersection over Union
FP	False Positive
FN	False Negative
TP	True Positive
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IVOD	In-Vehicle Object Detection
SVM	Support Vector Machine
RoIPooling	Region of Interest Pooling
RPN	Region Proposal Network
FEN	Feature Extraction Network
R-CNN	Region-based Convolutional Neural Network
YOLO	You Only Look Once
SGD	Stochastic Gradient Descent
RMSprop	Root Mean Square propagation
Adam	Adaptive Moment Estimation

1

INTRODUCTION

Since the start of the field of Artificial Intelligence that the ability to make computers perceive the world visually, as we do, has been a highly sought-after property of computer systems. In humans, the sense of sight is with little doubt our most important and we rely on the information that it provides more than any other sense.

Giving computer systems the same capability would be invaluable to increase their adaptability and allow them to operate on a better understanding of the world around it. Despite major progress in this area, we are still far away from simply giving a computer system a photo and have it fully understand the scene depicted and be able to use that information on its decision-making process the same way a human being would. Nevertheless the level of computer vision we currently have, still has important practical applications that continue to be explored as the field grows.

This project is concerned with the usage of computer vision and deep learning techniques to develop a system capable of detecting objects within a vehicle. Such a system could have a variety of uses in the car industry granting the on-board system a greater deal of information on the inside of the car and its contents. This could, in turn, be useful, for instance, for the driver as he could receive warnings about the contents of the car if there is anything of relevance in it as he prepares to leave, such as forgotten items for example.

Such a system, similarly to other machine learning projects, is not so much programmed as it is trained from data. And as such development of deep learning systems revolves mostly around two things. To acquire an appropriate dataset and to tweak the network to improve its detection performance as much as possible.

1.1 GOALS

As mentioned this project will be focused on the creation of an object detection system capable of detecting stray objects left within a vehicle. A Deep Learning system, more specifically a convolutional neural network, will be used to perform this detection due to their state-of-the-art detection performance in visual tasks such as object detection. The YOLO algorithm will be used as a basis for the analysis of the problem.

This thesis will be, specifically, concerned with the effect different architectures can have on the accuracy of object detection systems. To this end, several changes are iteratively made to the YOLOv3 architecture to better understand how these architecture design choices can affect the detection performance of the algorithm. We intend to study how well the YOLO architecture reacts to these changes and whether they help improve detection performance in the problem of in-vehicle object detection. In this way, this project will, potentially, shed some light on what techniques can better help improve convolutional neural network architectures.

1.2 DOCUMENT OUTLINE

This document is organized into five chapters as follows: The first chapter is the Introduction presenting the theme of this thesis. To that end, we define its context, motivation, and the objective it aims to achieve. Chapter two is a State of the Art, containing the result of the bibliographic search done as part of the early phases of this project to better understand its context and the work and research already done in its areas of study. Such fields of focus include computer vision, machine learning and deep learning and convolutional neural networks which are areas whose work can affect the development of this project. Following the bibliographic search, the searching strategy used to compile the bibliography is presented so as to better contextualize the results found during said search. In chapter three the objective of this thesis is delved into more deeply presenting some of the main issues that it intends to tackle and which are further explored throughout this document. It also presents some of the design decisions that were made as the basis for all experiments performed and which assumptions the rest of this thesis builds upon. Chapter four presents the experiments done and the results obtained from said experiments. Some analysis is also made into the possible significance of said results and what this could mean in relation to the objective of the thesis. Lastly, in chapter five, Conclusions and Future Work, the results of this thesis and its conclusions are compiled for further analysis and other future research ideas are presented.

2

STATE OF THE ART

The development of this project will focus primarily on the usage of deep learning to attempt to perform object detection within a vehicle. This section, presents an introduction to these topics and the important work previously done in this area, to better contextualize the problem.

2.1 BIBLIOGRAPHIC SEARCH STRATEGY

The bibliographic search upon which this project is based was done mostly through the usage of bibliographic search engines such as Semantic Scholar and Google Scholar. This search went through a cycle, alternating several times between a phase of actual search, in which a list of articles of possibly relevant was compiled, and a phase in which the relevance of each article was better investigated, removing irrelevant articles. The initial search was performed using a combination of search terms including ”computer vision”, ”convolutional”, ”neural networks”, ”object detection” and ”object recognition” but analysis of the results led to a refining of the search giving preference to more recent articles particularly since 2012. Preference was also given to articles with high Citation Velocity as for recent articles this can be a good measure of article performance. Once finished the search, reading of the Abstract, Introduction and Conclusion of the papers gave a better idea of the importance of each article, in this way some articles were removed from the list as they proved irrelevant to the topic at hand while others were given higher priority in the to-read list for their importance. A look at the references section in some of these articles also revealed some interesting papers that were then added to this list for further analysis.

2.2 COMPUTER VISION

Computer Vision is a field of computer science dedicated to giving computers the capability of sight. More specifically its main objective is to allow computers to derive high-level understanding from visual information such as pictures and video(Learned-Miller, 2011). Despite the apparent simplicity of seeing, given how easily human beings do it on a daily basis, allowing computers to do so on a similar level has been an incredibly complex and difficult task and is still an ongoing topic of research since its inception

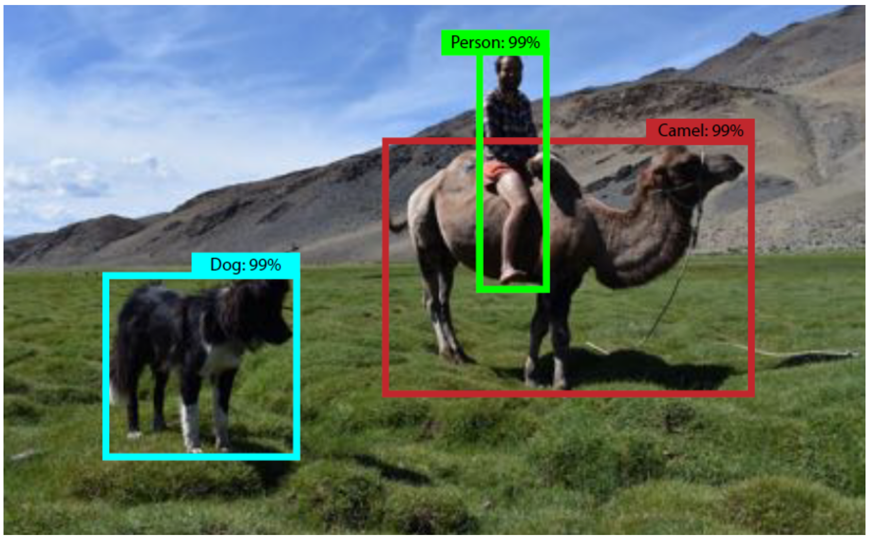

Figure 1: Example of the output of an object detection algorithm with bounding boxes around each object present in image with respective classification and the confidence of the algorithm on the result, adapted from (Redmon & Farhadi, 2018)

in the 1970s[1](Szeliski, 2010). Given the somewhat nebulous definition of vision, research in this area usually concerns itself with smaller and better-defined subproblems such as object recognition(Lowe, 1999), 3D pose estimation(Murphy-Chutorian & Trivedi, 2009) and video tracking(Yilmaz, Javed, & Shah, 2006).

This project is particularly concerned with the task of object detection which, in the context of computer vision, means trying to find every object in a picture, returning bounding boxes and a classification for each of them. This classification is one of several object classes defined in the problem statement, given which types of objects have to be identified. Because of this, object recognition and localization, which attempt to classify and generate bounding boxes, respectively, for the single main object in a picture, are also of interest as techniques utilized in these areas can often transfer well to detection problems(Sermanet et al., 2013). In recent years it has become apparent that Machine Learning can be a profoundly useful tool to computer vision allowing researchers to make effective use of the increasing amount of data, without the effort associated with the hand-engineered tools that used to be so common in the field.

2.3 MACHINE LEARNING

Machine Learning is the study and development of algorithms that allow a computer to learn. Learning is defined as the ability to recognize patterns in a set of data that can then be used to make predictions, such as classifying similar, but never seen before, data instances or predicting the results of a certain

1 there is some small work done on this subject as early as 1966, but it can be considered that the field came into its own with the more serious computer vision research done in the early 1970s

event(Mitchell, 1997). These algorithms often work by using statistical techniques to implicitly[2] estimate the probability distribution of the data(Goodfellow, Bengio, & Courville, 2016).

Machine Learning approaches are usually divided between Supervised learning and Unsupervised learning. In Supervised learning, some sort of indicator is provided by the user to direct the algorithm towards what exactly is meant to be learned. This indicator frequently takes the shape of labels which are the value that the algorithm must learn to predict for each of the training examples. On the other hand, in the case of Unsupervised learning no such indicator exists, and therefore the learning algorithm merely captures patterns in the data that are meant to be relevant according to a variety of metrics including, for instance, how often this pattern shows up in the dataset. It is then up to the user to attempt to make the most out of these patterns and apply them where they are most useful.

There are also other important sub-groups that are important enough to be worth mentioning for a good overview of the field such as Reinforcement learning and Semi-Supervised learning. Reinforcement learning, unlike other approaches, is applied to dynamic environments and provided with a *reward* signal that tells the algorithm how well it is performing. The algorithm must then learn how to act in order to maximize its future reward. Semi-Supervised learning is similar to Supervised learning but only a subset of the dataset is labeled, the rest is unlabeled(Goodfellow et al., 2016).

One of the main problems that Machine Learning tries to tackle is the problem of generalization, or in other words, whether the patterns learned from the data apply to the new examples it will possibly see in the future. If a model does not generalize well, and the learned patterns only apply to the data already seen then it can not be said that it actually learned anything particularly useful. In order to properly evaluate the capability of the model to generalize it is frequent to divide the dataset between training set and test set. The learning algorithm tries to create a model as accurate as possible based on the data on the training set but without any information about the test set. Then the model is used on the test set to get an idea how well it would behave on new previously unseen data but the model itself is not changed during this process, in other words, it does not try to learn from the test set.

Two metrics of system detection performance come out of this training paradigm, training error and test error. High training error, also know as underfitting, shows that the learning algorithm is missing the relevant patterns in the data. This means that the learning algorithm is just not working, and it might be necessary to fundamentally change it somehow, though what the possible changes are, depends on what algorithm is being used. One possible cause that could lead to underfitting is using a model whose capacity is too small. The capacity of a model is its ability to correctly approximate a wide variety of functions. Often increasing the capacity of a model can result in training error improvements (Goodfellow et al., 2016).

While what truly matters in Machine Learning is the test error, or how well the model generalizes, the test error is at best as low as the training error so minimizing the latter can be just as important as minimizing the difference between the two. When this difference is high (when the test error is much higher than the training error) it is said the model has overfitted. This means that the learning

2 or explicitly in the case of density estimation

algorithm has learned the details of each specific training example rather than the overall patterns that are common to the whole dataset. This severely limits the capability of the model to generalize which leads to the elevated test error. Usually the most straightforward way to fix this is by getting more training data, unfortunately, this is not always viable. Another common solution is early stopping, where, to stop the algorithm from overfitting, the training process is stopped when the test error starts decreasing. While what has been said throughout this section applies to most Machine Learning algorithms, this project will focus mostly on artificial neural networks a more specific learning system that will be explored in more detail in the next section (Goodfellow et al., 2016) (Domingos, 2012).

2.4 DEEP LEARNING

Artificial Neural Networks (ANN) are a Machine Learning system loosely inspired by the biological neurons that can be found in our brains. These systems are represented as a series of units, called neurons, divided into layers. Each neuron computes a non-linear function of its inputs, giving different weights to each one, and communicates it to the following layer. At the end of the network the *loss* is calculated, which in supervised learning, is usually some measure of the difference between the expected and actual output. Afterward, the backpropagation algorithm is used to calculate the gradient of the loss with respect to each of the weights. And, depending on the optimizer that is used, the weights are updated to their new value for the next iteration.

Frequently used optimizers are Stochastic Gradient Descent (SGD), Root Mean Square Propagation (RMSprop) and Adaptive Moment Estimation (Adam) (Kingma & Ba, 2014). This process repeats, iteratively minimizing the loss and, in doing so, finding the model parameters (weights) that best capture the intended distribution. Each iteration of the algorithm through the entire training dataset is called an epoch. A common practice when training neural networks is transfer learning where a network is trained with the data for a different but related problem before actually starting the training with the real data. The first training is called pre-training while the training with the data for the real problem is called fine-tuning. It has been shown that transfer learning can be incredibly helpful, often needing much smaller datasets, as when fine-tuning the network can make use of the parameters already learned during pre-training taking much less time to minimize the loss (Yu, Deng, & Dahl, 2010) (Erhan et al., 2010).

One of the first examples of ANNs appeared in 1958 under the name *perceptron* (Rosenblatt, 1957) as an implementation of Hebbian learning, a neuroscience theory of how the brain could possibly be learning. At the time these networks had several limitations compared with their more modern variants. For instance, one of the most famous cases is the incapability of ANNs of the time, to learn the exclusive-or function (Minsky & Papert, 1969). Despite small advancements throughout the years (LeCun et al., 1989) (Rumelhart, Hinton, & Williams, 1986), have remained relatively small in the research world until very recently with the advent of deep learning.

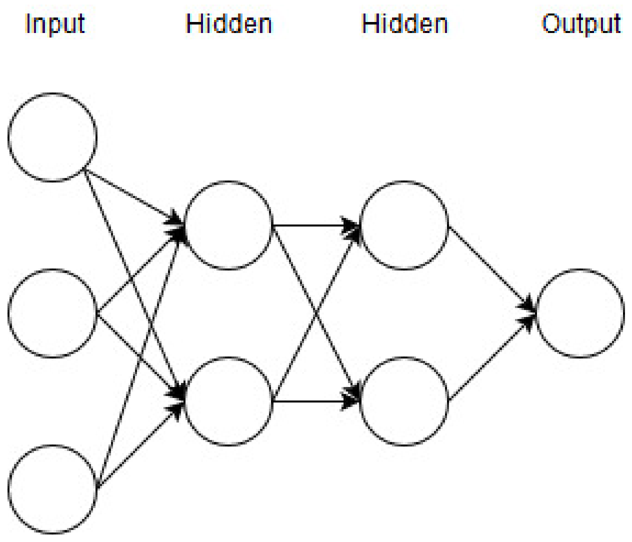

Figure 2: Artificial Neural Network with 2 hidden layers

Deep Learning refers to the capability of some machine learning algorithms (most commonly ANNs) to stack several layers of processing units in order to learn different, successively more abstract, representation for the data. While this idea was talked about as early as 1986 (Dechter, 1986) implementing it effectively took much more data and computational power than was available at the time. Recently, due to both the enormous amount of data made available thought the internet and the increasing computational power of graphics cards, whose optimization for parallel computation makes them ideal for ANN computations, it became practical to implement this idea. In 2006 the term ”Deep Learning” started gaining popularity after Geoffrey E. Hinton, Simon Osindero and Yee-Whye Teh showed that a deep belief network could be better trained by pre-training each layer individually before fine-tuning (Hinton, Osindero, & Teh, 2006). Deep Learning has since become a major area of research, dramatically improving state-of-the-art in computer vision, speech recognition and natural language processing (Goodfellow, Bengio, Courville, & Hinton, 2015).

2.5 CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNN) are a specific type of Deep Neural Networks that feature some layers that make them especially suited for working with visual input. More specifically, there are the convolutional layers and the pooling layers. The convolutional layers make use of the convolution operation to apply a filter to the entire image. One of the advantages of these layers over fully connected layers is the reduced number of parameters. In a fully connected layer, the number of parameters is dependent on the size of the input and the size of the layer which in the case of images would mean more than a parameter per pixel of an image. This would translate into longer training times and more chances of overfitting. Convolutional layers, on the other hand, are independent of input size, the number of parameters in the layer being dependent only on the size of the filters and the number of filters. As such training is easier and the model gives reduced importance to every single pixel when compared to filter-sized chunks of the picture which makes sense as in practice very rarely does a single pixel carry much weight on the overall information imbued in the image.

Pooling layers allow the efficient reduction of the size of the representation of the data which means the network uses fewer parameters. The most common forms of pooling are max-pooling and average-pooling. In max-pooling, a filter is applied throughout the image returning only the max value it captures. On the other hand, average-pooling returns the average of the values caught by the filter. The most common CNN architectures tend to be composed of an alternated series of convolutional layers followed by the pooling layers with a few fully connected layers at the end of the network. These two types of layers in a mostly alternated fashion followed by a few fully-connected layers are the most common CNN architecture. Each successive convolutional layer is responsible for detecting more complex and abstract feature of the image from the simpler features captured in previous layers (Zeiler & Fergus, 2014).

With the rise of deep learning, it became apparent that ANNs could be used to achieve unexpectedly good results in a variety of difficult problems given enough data. In 2012, AlexNet (Krizhevsky, Sutskever, & Hinton, 2012) was the winner of the ImageNet (Deng et al., 2009) classification task showing that deep convolutional neural networks could be used to greatly improve state-of-the-art detection performance with a top-5 error rate of 15.3%, an advantage of 10 percentage points over the second place contestant which did not use CNNs. This CNN had 7 hidden layers and 60 million parameters, making use of a variety of techniques that would become common in a variety of other CNNs such as max-pooling, ReLU non-linearity and a GPU implementation to speed up the training. From then on CNNs have been consistently used to try to solve Computer Vision problems.

In fact, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an extremely influential Computer Vision competition delivering many important ideas that have improved state-of-the-art detection performance. One such example is the VGG networks (Simonyan & Zisserman, 2014) which won the ILSVRC2014 for classification and localization. They made popular the idea of using small filters (3x3) instead of bigger ones like the ones used in AlexNet (11x11). In their introductory paper, several different networks were compared with similar architecture but increasing depth to test its effect on detection performance. The one that performed better was also the deepest model with 16 convolutional layers, 5 max-pooling layers and 3 fully connected layers. Both VGG and AlexNet networks are often used in other papers as pre-trained image classifiers (Han, Zhong, Cao, & Zhang, 2017) (Chen, Papandreou, Kokkinos, Murphy, & Yuille, 2014).

Other interesting examples include GoogLeNet (Szegedy et al., 2015) and ResNet (He, Zhang, Ren, & Sun, 2016). GoogLeNet introduced the inception module which uses several parallel convolutional layers with different filter sizes. The idea behind the inception module is that, rather than having to pick the best filter size for the problem, the developer can leave that decision to the learning algorithm. As each path has different filter sizes the learning algorithm takes care of giving bigger weights to the path

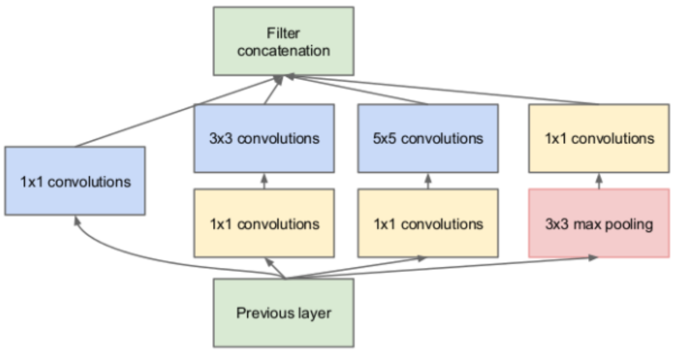

Figure 3: Inception Module, adapted from (Szegedy et al., 2015)

with the filter size that better fits the problem. Unfortunately, these alternate paths create the problem of increasing considerably the number of parameters of the model which in turn greatly increases the computational resources necessary to train and use the model. In order to alleviate this problem, the paper uses 1x1 sized convolutional layers to reduce the number of features before going through the convolutional layers with bigger filters. This greatly reduces the number of operations necessary to train the network while maintaining state-of-the-art detection performance. The Inception Modules used in GoogleNet can be observed in Figure 3.

In the ResNet a new powerful idea was implemented, to add shortcut non-weighted connections across layers, allowing the network to easily learn the identity function of the input if the extra parameters prove unnecessary for the problem at hand. This innovation helps reduce the exploding/vanishing gradient problem and allowed the ResNet to win first place in the ILSVRC2015 classification, localization and detection tasks. The vanishing gradient problem happens when as ANNs become deeper the gradients used to update the weights of the network during training keep getting smaller throughout the network slowing the training to a crawl. The exploding gradient problem is the opposite problem when the gradient becomes too big potentially causing overflows. With the shortcut layers, it becomes possible to have deeper networks than it was possible to have before.

One of the most important recent advancements in object detection is the Region-based Convolutional Neural Networks (R-CNN) (Girshick, Donahue, Darrell, & Malik, 2014). Up until this time object detection was mostly done through a sliding window approach, but in R-CNN selective search (Uijlings, van de Sande, Gevers, & Smeulders, 2013) is used to apply bounding box candidates in the image, finding around 2000 region that proposals that are most likely to contain objects. This ends up increasing the detection performance of the algorithm considerably when compared to state-of-the-art sliding window algorithms. These region proposals are then put through a pre-trained AlexNet which extracts the most important features of the region for classification. Finally, these features are used by Support Vector Machines (SVM) to classify the images. While this approach was very successful at the time, gaining an improvement in the state-of-the-art detection performance of over 10% in the Pascal VOC

2010 dataset and 7% in ILSVRC2013, this approach was still very inefficient and several improvements have since been released that attempt to make it faster.

In ”Fast R-CNN” (Girshick, 2015) Region of Interest Pooling (RoIPooling) layers are used to allow the R-CNN to receive input of various sizes. This is achieved by using a variable size filter that changes depending on the size of the input to guarantee the same output size and using global features previously extracted from the dataset only once. The ”Faster R-CNN” (Ren, He, Girshick, & Sun, 2015) further improves on the ”Fast R-CNN” by dividing the object detection process into three phases. Firstly there is the Feature Extraction Network (FEN) which gets the important features from the images for the next phase. These features are fed into a region proposal network (RPN) (Szegedy, Reed, Erhan, & Anguelov, 2014) which replace selective search as the region proposal method while being much faster. The RPN is trained to predict regions based on the previously extracted features, and several reference boxes, known as anchor boxes. These region proposals are then used as input by another CNN that predicts the bounding box the object in that region and classifies it, namely the Classification Network. The idea is that using a CNN with anchor boxes to generate the region proposals will be much quicker and more accurate than selective search.

The You Only Look Once (YOLO) (Redmon, Divvala, Girshick, & Farhadi, 2016) is an object detection algorithm that, while quicker than the Faster R-CNN, is less accurate. Unlike the R-CNN approach which separates the detection process in two phases, YOLO is end-to-end, receiving an image as input subdividing it in a 7x7 grid and attempting to classify and give bounding boxes for the object whose center is in each of the cells in the grid. This makes it not only much faster, achieving real-time detection, but also easier to optimize as improvements can be made to the whole process, instead of optimizing localization and classification separately. On the other hand, YOLO loses quite a bit of detection performance for that increase in speed scoring about 10 percentage points lower than ”Faster R-CNN” in Pascal VOC 2007. Several improvements were implemented in YOLOv2 (Redmon & Farhadi, 2017) allowing it to reach state-of-the-art detection performance while still maintaining real-time detection. YOLOv2 like ”Faster R-CNN” uses anchor boxes to better predict bounding boxes but unlike the latter, the anchor box shapes are learned used k-Means from data. Other improvements used are batch-normalization, multi-scaling the images for training, removing the fully-connected layers making the network entirely convolutional and smaller. YOLOv2 manages 73.4 mAP when training on Pascal VOC 2007 and 2012. Given the importance of the YOLO algorithm for this thesis, it will be explored in further detail in future chapters.

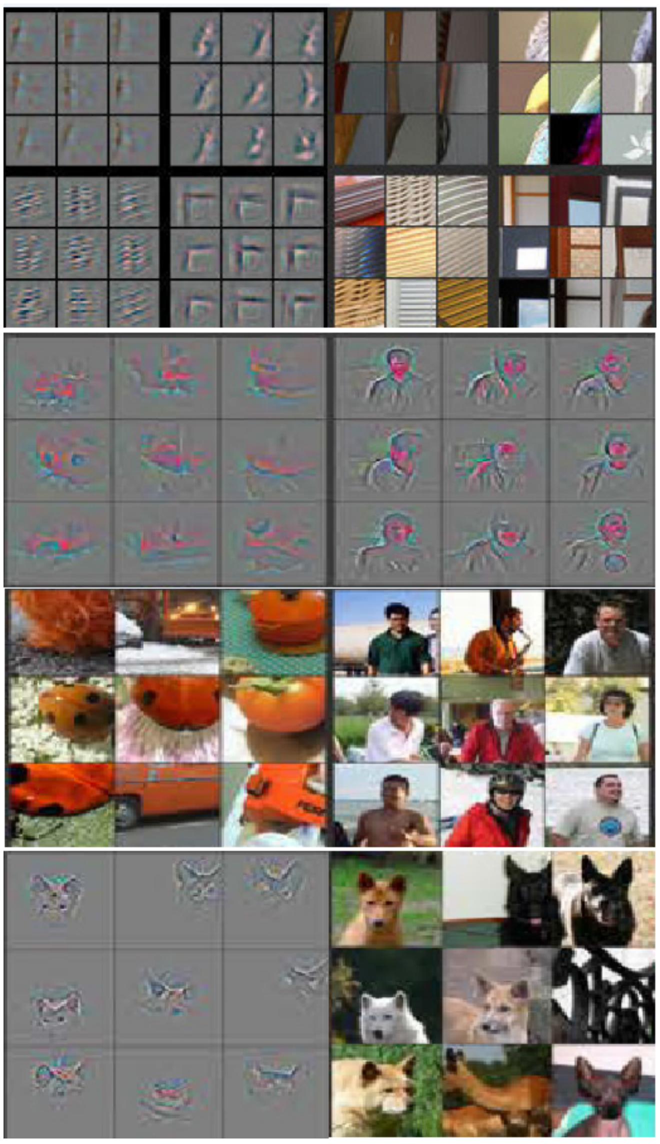

Figure 4: Visualizing Learned features in a CNN and examples of images that maximize neuron activation in different layers, adapted from (Zeiler & Fergus, 2014)

2.6 RELEVANT LITERATURE

Table 1: Relevant Literature

Reference	Area of Interest	Contribution
Learned-Miller (2011) Szeliski (2010)	Computer Vision	Books that introduce the field of computer vision, its important topics and advances made in trying to solve its problems
Mitchell (1997) Domingos (2012)	Machine Learning	Material that presents the fundamentals of Machine Learning
Kingma and Ba (2014) Rosenblatt (1957) Minsky and Papert (1969) LeCun et al. (1989) Rumelhart et al. (1986) Dechter (1986)	Deep Learning	Kingma and Ba (2014) developed the optimization algorithm ADAM Rosenblatt (1957), Minsky and Papert (1969), LeCun et al. (1989), Rumelhart et al. (1986), Dechter (1986) are historical references as related to the rise of Deep Learning
Hinton et al. (2006) Goodfellow et al. (2015) Goodfellow et al. (2016)	Deep Learning	Hinton et al. (2006) introduced a new method of training networks that many considered to be one of the main reasons for the renewed interest in neural networks that gave birth to Deep Learning Goodfellow et al. (2015), Goodfellow et al. (2016) review and book, respectively, exploring the field of Deep Learning
Krizhevsky et al. (2012) Simonyan and Zisserman (2014) Szegedy et al. (2015) He et al. (2016)	CNNs for Classification	Krizhevsky et al. (2012) (AlexNet), Simonyan and Zisserman (2014) (VGG nets), Szegedy et al. (2015) (Inception), He et al. (2016) (ResNet) each introduced new architectures that changed the current state of the art CNN performance making different techniques standard practice in the design of CNNs

Girshick et al. (2014) Girshick (2015) Ren et al. (2015) Redmon et al. (2016) Redmon and Farhadi (2017) Redmon and Farhadi (2018)	CNNs for Detection	Girshick et al. (2014) (R-CNN), Girshick (2015) (Fast R-CNN), Ren et al. (2015) (Faster R-CNN) introduce and improve R-CNN algorithm, revolucionary object detection algorithm that greatly pushed state of the art performance forward using CNNs Redmon et al. (2016) (YOLO), Redmon and Farhadi (2017) (YOLO9000), Redmon and Farhadi (2018) (YOLOv3) introduce and improve the YOLO algorithm which use single shot detection to achieve real-time inference while remaining competitive with state of the art performance

3

PROBLEM AND ITS CHALLENGES

With the rise of Deep learning as an active area of research, it has become clear how important it is to further study this area and to increase our understanding of DNNs. The advances recently achieved in this area have already resulted in severe improvements to a variety of areas including computer vision, and it is very likely that further research in this field of study will result in even more developments in the future. Given the results already achieved in computer vision and, at the same time, our still limited understanding of many facets of the inner working of DNNs further insights about this topic could greatly enhance computer vision.

The aim of this thesis is the development of a system capable of detecting stray objects within the cockpit of a vehicle. This system will be expected to be able to not only find the location of objects inside the car but also classify them within a series of pre-defined classes. These classes were chosen based on what one is likely to find inside a car such as jackets, cell phones, and umbrellas. As such this problem falls within the scope of the object detection task of computer vision. This means that a main focus of the development will be attempting to adapt the several available approaches to solving this task to this specific problem and seeing how well they perform.

Right away it becomes apparent that one fundamental difficulty with this project is going to be the lighting as, dependent on a variety of circumstances, from time of day to location and weather, different amounts of illumination are going to be available, at different times, potentially making the task much harder. At night, for instance, the inside of the car can often become enshrouded in darkness making any attempt at detecting objects all but impossible. Therefore special care will have to be taken to figure out what can be done to improve this situation. Data Augmentation is one approach that is used to increase the detection performance of the system in a range of different lighting conditions.

Given the difficulty and the time cost of designing a new ANN architecture from scratch, it is common practice to start real-world applications using an already mature and well-tested architecture as a starting point. Starting with the YOLOv3 algorithm as a baseline various modifications are made to try to improve the detection performance of the algorithm. One key idea that is explored in detail throughout the experiments is the idea of parallelism. As shown by the results achieved by GoogleNet with the Inception Module (Szegedy et al., 2015) it is apparent that the use of parallel layers rather their sequential use could help make the algorithm better.

While several of these problems can be partially solved by a correct choice of dataset, that can be a problem in itself as acquiring a sufficiently large and diverse dataset for a specific problem can quickly become impractical. This being said measures can and should be taken to either make this acquisition easier or less necessary by using other solutions to try to fix the problems faced by the system. It is also easy to unintentionally make a dataset that is not an accurate portrayal of the problem under study. Because of this, the continued improvement of deep learning algorithms is of the utmost importance so as to make them more capable of learning in a variety of circumstances and with less than ideal data availability.

3.1 DATASETS

Two datasets were used in this thesis, the In-Vehicle Object Detection (IVOD) dataset, and the smallCOCO dataset.

The IVOD dataset is an image dataset containing images of the inside different vehicles with a variety of objects strewn around the cockpit. IVOD is a proprietary dataset collected while working with Bosch Car Multimedia.

Common locations for objects include the driver and passenger seats and the ground. Five cars were used for background in different locations to provide a wide variety of lighting conditions. The different cars are worthy of mention as it was essential, given the context of the problem, to make sure the algorithm did not overfit to the background of the particular car in use. If the algorithm overfitted in this manner it would have resulted in a less useful detection system as it is intended to be used in an array of vehicles. Using multiple cars makes the algorithm more robust to changes in the background and the car itself.

There are 21 object classes that can be found in the vehicles and the distribution of these classes can be found in Table 2. These classes were chosen based on what seemed to be the most common items one can find lying around a car cockpit. The camera for collecting the photographs was placed behind the front seats in the middle of the ceiling pointing to the front of the car and slightly downwards.

During the collection, objects were put into the car in such a manner as to simulate real conditions that can be found in daily life. For example, an example that can often be found in the real-world are objects left on the ground, such a location can present a serious problem for the algorithm as natural lighting can often have trouble reaching the ground making detection more difficult. Another common example is that of overlapping objects which might make the task of separating the two objects and classifying them appropriately, a complex task. Both of these examples were taken into account when collecting the dataset so as to give more realistic data for the algorithm to train on. An example of the kind of image contained in the IVOD dataset can be seen in Figure 5.

The smallCOCO dataset is a subset of the COCO dataset (Lin et al., 2014) modified to be smaller in scope for quicker testing as the original COCO dataset took too long to converge for the testing purposes of this thesis.

Table 2: The classes of the IVOD dataset

coin	135
pen	146
watch	152
cellphone	1652
glasses	1014
scarf	429
gloves	183
backpack	885
mouse	83
coat	1337
computer	170
cap	108
umbrella	464
notebook	1582
keyboard	8
key	337
card	238
bottle	316
bill	46
purse	308
wallet	1516
Total Dataset Size	3197

The COCO dataset (Lin et al., 2014) is one of the major object detection datasets and is often used as a metric of detection performance by many object detection algorithms. It contains 80 classes of common objects in daily life conditions in a variety of settings. In total the dataset is comprised of 476688 images, 456567 images for the training set and 20121 for the validation set. This dataset presents a considerable amount of variability presenting all manner of objects, backgrounds, lighting conditions and color palettes.

In the smallCOCO dataset the number of classes was altered from the original 80 to 11 and the number of images was also greatly reduced. The distribution of these classes in the smallCOCO dataset can be found in Table 3. This dataset ends up with 30401 images with 20387 images in the training set, 7000 images in the validation set and 3014 in the test set. Despite the reduction in the size of the dataset it still remains bigger and more complex than the IVOD dataset and presents many locations beyond cars.

This is not particularly important to the main problem of this thesis, specifically object detection within vehicles, and can even be detrimental as using it means training the algorithm for an entirely different problem than the one under study. On the other hand, this makes it ideal for the few occasions in which it is valuable to find out how applicable the results of the experiments are to more general object detection. And indeed the dataset does end up being used for that purpose.

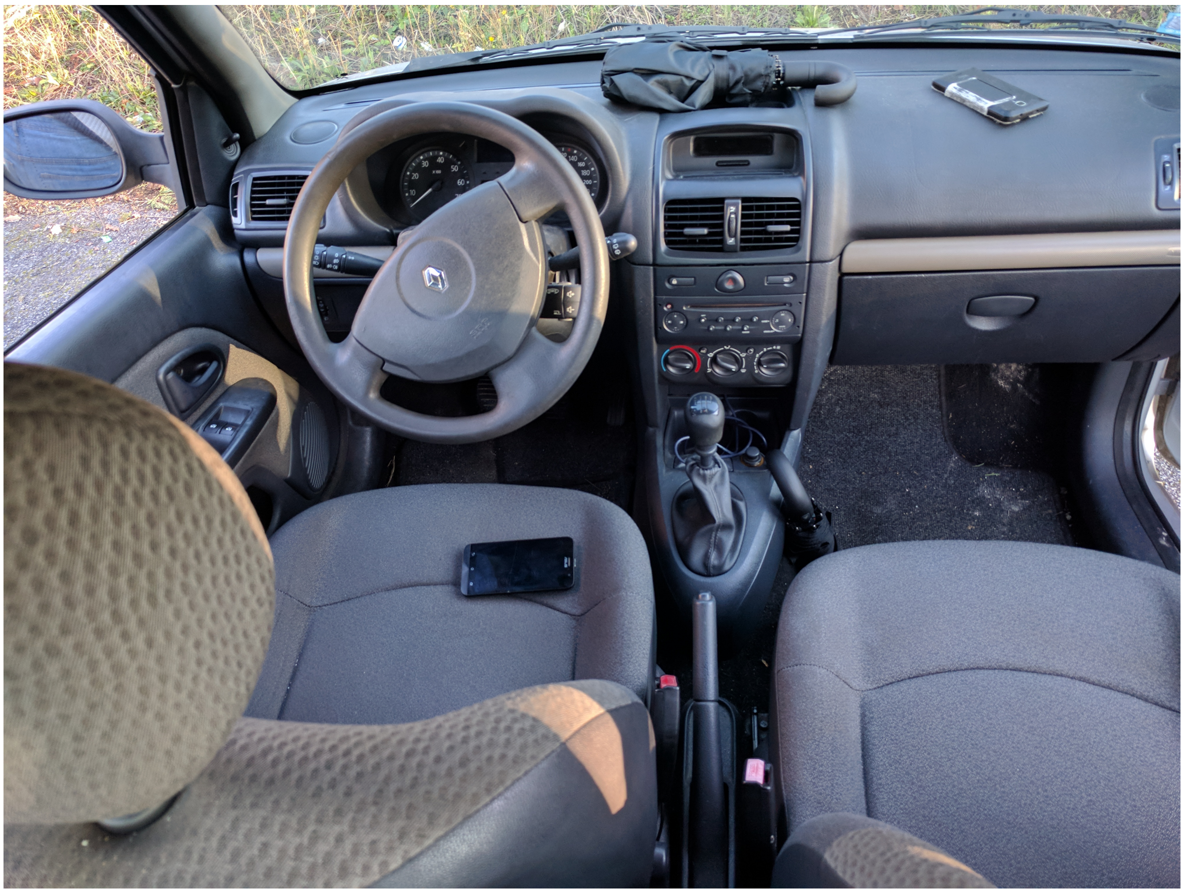
Figure 5: Example picture from the IVOD dataset

Table 3: The classes of the smallCOCO dataset

backpack	9091
umbrella	38244
handbag	12896
tie	6767
suitcase	6524
laptop	5200
mouse	2367
remote	5987
keyboard	3006
cellphone	6701
book	26322
Total Dataset Size	30401

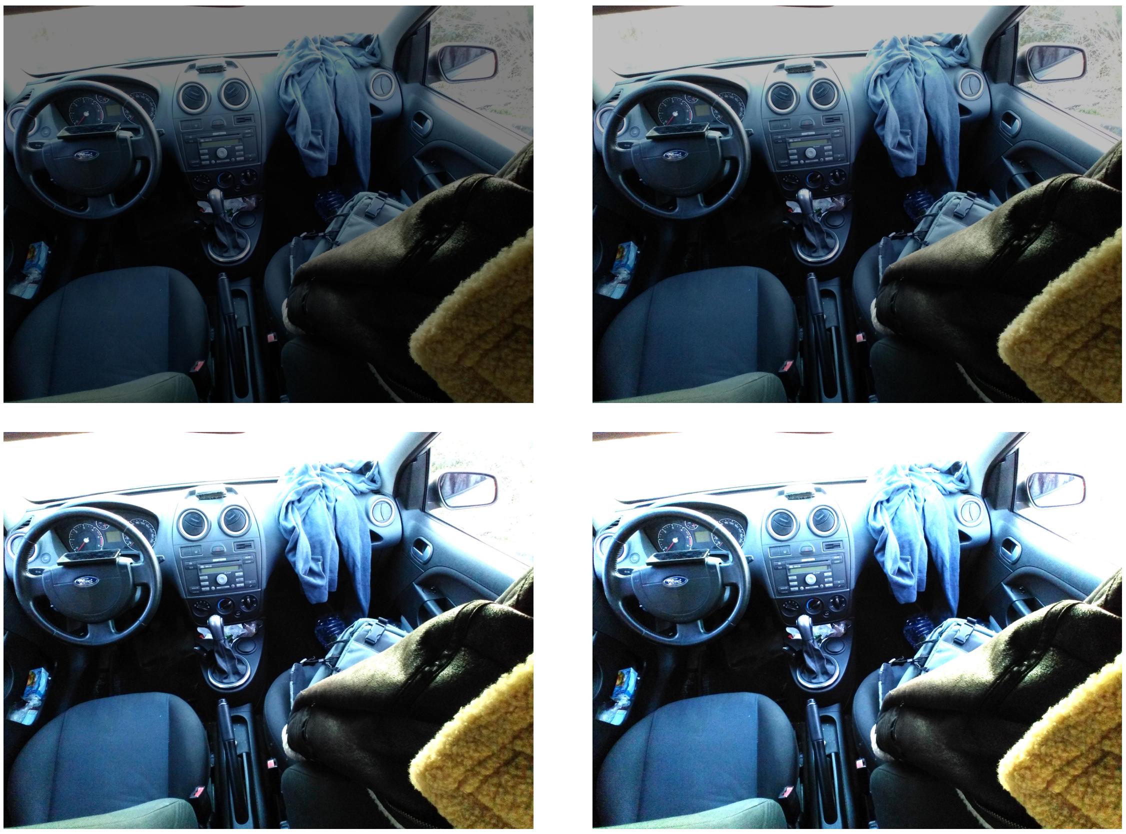
Figure 6: Examples of augmented images

3.2 DATA AUGMENTATION

Given the relatively small size of the IVOD dataset and the importance of different illuminations, data augmentation was used by changing the brightness of the pictures in an attempt to simulate images with better or worse lighting. For each picture in the dataset four other copies of images were created with respectively 50%, 75%, 125% and 150% the original brightness. As such the first two images resulted in a darker image than the original helping to prepare the algorithm to deal with darker images such the ones it might find in darker settings like an overcast day, when the car is in the shade or during the night. On the other hand, the remaining two images are brighter than the original photograph better preparing the algorithm for brighter settings such as when the car under direct sunlight. Examples of augmented images can be found in Figure 6.

3.3 THE MAP METRIC

The mAP, or mean Average Precision, is one of the most commonly used accuracy metrics in object detection and is used as the main metric by the YOLO authors (Redmon et al., 2016) and the R-CNN authors (Girshick et al., 2014). It is also used as the main metric in the major detection challenges and datasets such as the ImageNet detection challenge (*Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) Results*, n.d.), and the COCO Object Detection Task (*COCO Detection Leaderboard*, n.d.).

In order to better understand the mAP metric, it is necessary to first understand a few its underlying concepts. Firstly it is crucial to understand how the correctness of a detection is evaluated. A detection is considered correct, also know as a True Positive (TP), if both the class of the prediction and its bounding box match the ground truth presented in the label of the image. For the purposes of the mAP two bounding boxes are considered a match if they have an Intersection-over-Union (IoU) of over 0.5.

By comparing the predictions given by an algorithm to the ground truths the precision and recall can then be calculated. Precision is a measure of the number of predictions that do not in fact correctly correspond to any of the ground truths, also commonly called False Positives (FP), and can be calculated with the following formula:

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

Recall is used to measure the number of False Negatives (FN), ground truths that were not detected at all, and can be calculated using the following formula:

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

Given these two values, the mAP is calculated by computing the Average Precision (AP) for each class and averaging them out. The AP is the average of the maximum precision for various recall values.

3.4 THE DARKNET FRAMEWORK AND THE YOLO ALGORITHM

All the experiments were done using the Darknet framework (Redmon, 2013–2016). The choice of which framework to use was an important one, early on, as it would affect the entire experimental process. The right framework could greatly accelerate the development and testing of new architectures. The Darknet was chosen as it was the framework in which the YOLO architecture was implemented. The YOLO architecture was chosen because, unlike most of its competitors, it managed to do real-time inference. The speed at which the YOLO algorithm is able to do inference was a key deciding factor in it being chosen since quick inference was deemed a major factor in the applicability of the system in real vehicles.

Originally an implementation of the algorithm in Tensorflow called Darkflow (*Darkflow Github*, n.d.) was considered but with the release of the new version of the algorithm, YOLOv3 (Redmon & Farhadi, 2018), the decision was made to stick with the original Darknet to take advantage of the new algorithm as Darkflow was not being updated and as such would not implement YOLOv3. The experiments performed, use the YOLO architecture as a starting point and as alterations are made to the architecture, tests are done to measure the effect these variations had on algorithm detection performance.

Initially, the YOLOv2 architecture (Redmon & Farhadi, 2017) was going to be used as the basis for the experiments but around the start of this thesis the new YOLOv3 architecture (Redmon & Farhadi, 2018) was released. As such a comparative test was done between the two architectures to better measure the improvements made to the new algorithm. This test is presented in the next chapter. Given the better detection performance of the YOLOv3 algorithm, it was used as the basis for the tests throughout this thesis. The YOLOv3 architecture is explored in detail in the following section.

3.5 YOLOV3 NETWORK ARCHITECTURE

YOLO is an object detection algorithm focused on speed, while it does not always match state of the art detection performance it makes up for it by being quick. A comparison between a few important object detection algorithms can be found in Table 4. It achieves this by dividing the image into cells of the same size with a grid and attempting to predict bounding box and classifications for each grid cell as a regression problem. Unlike the multi-stage process other object detection algorithms like R-CNN use, the YOLO algorithm just provides the images to a single ANN which then outputs the predicted classification and the coordinates for the bounding boxes for each grid cell.

The YOLO algorithm makes use of 9 anchor boxes to help it correctly predict the dimensions of its bounding boxes. The YOLOv3 also lacks any fully-connected layers which allows it to function with several distinct image sizes. The higher the resolution of the input images, the better the accuracy achieved by YOLO, with the catch of making it slower as can be seen in Table 4. The YOLOv3 also lacks pooling layers which helps it detect smaller objects which is a problem that tends to affect the various YOLO algorithms. Instead, it alternates the convolutional layers with shortcut layers which have been in ResNet (He et al., 2016) to help improve detection performance. The convolutional layers in YOLOv3 use leaky ReLU as their activation function and use batch normalization to avoid overfitting.

Another key feature of the YOLOv3 is the usage of a concept similar to feature pyramid networks (Lin et al., 2016) to use higher resolution features from earlier in the network so the network can both use the lower resolution features for more general information and higher resolution features for more finer-grained information. This system can be observed in the Table 5. After the main feature extractor, the feature map of 13x13 is fed to the first detection layer. Following this, the same feature map is upsampled to a bigger size and then combined with a feature map from earlier in the network by the route layer resulting in a feature map of 26x26. This feature map ends up in another detection layer after going through several convolutional layers. This whole process is repeated once more after this detection to get a 52x52 feature map. For ease of reference, this mechanism is referred to as paths throughout this thesis. Each path refers to what layers the data goes through before arriving at a detection (output) layer. As such the YOLOv3 has 3 paths each with successively more resolution feature maps than their predecessors.

Table 4: YOLOv3 detection performance compared to other Neural Networks, adapted from (Redmon & Farhadi, 2018)

Network	mAP	time (ms)
SSD321	28.0	61
DSSD321	28.0	85
R-FCN	29.9	85
SSD513	31.2	125
DSSD513	33.2	156
FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Table 5: The YOLOv3 architecture

	Layer	**Filters**	**Filter size**	**Stride**	**Activation**
	Convolutional	32	3 x 3	1	ReLU
	Convolutional	64	3 x 3	2	ReLU
	Convolutional	32	1 x 1	1	ReLU
1x	Convolutional	64	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	128	3 x 3	2	ReLU
	Convolutional	64	1 x 1	1	ReLU
2x	Convolutional	128	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	256	3 x 3	2	ReLU
	Convolutional	128	1 x 1	1	ReLU
8x	Convolutional	256	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	512	3 x 3	2	ReLU
	Convolutional	256	1 x 1	1	ReLU
8x	Convolutional	512	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	1024	3 x 3	2	ReLU
	Convolutional	512	1 x 1	1	ReLU
4x	Convolutional	1024	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	512	1 x 1	1	ReLU
3x	Convolutional	1024	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	256	1 x 1	1	ReLU
3x	Convolutional	512	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	128	1 x 1	1	ReLU
3x	Convolutional	256	3 x 3	1	ReLU
	Detection	—	—	—	—
#layers	106				

3.6 TRAINING STOPPING POINT

When training machine learning systems the point at which the training should be stopped should be carefully considered. On the one hand if training is stopped too early then the system will have been unable to fully learn all the worthwhile information in its dataset performing worse than it could have otherwise. On the other, if the system is allowed to train for too long it could end up overfitting to the data deteriorating its generalization ability and compromising its accuracy with data other than its training set. At regular points during the training, the network accuracy on its validation set was measured to get an understanding of the learning progress. The mAP achieved at these points in the training were then used to compile graphs that can be seen in the Results chapter. After several experiments, it became obvious that 20000 iterations was a good stopping point and so the experiments done in this thesis contain data only up to that point.

3.7 ENVIRONMENT DESCRIPTION

All the experiments were done on a computer with the specifications presented in Table 6.

Table 6: Training machine specifications

CPU	Intel Core (TM) i7-7800X
CPU clock speed	3.50GHz
RAM	64GB
GPU	Nvidia GTX 1080 Ti 11GB
CUDA	CUDA Version 9.0.176
OS	Ubuntu 16.04.4 LTS

4

EXPERIMENTAL RESULTS

This chapter will present the experiments performed throughout this thesis. All the experiments were done using the Darknet framework (Redmon, 2013–2016). The experiments performed start with the YOLO architecture and as alterations are made to the architecture, tests are done to measure the effect these variations had on algorithm detection performance. Initially the YOLOv2 architecture (Redmon & Farhadi, 2017) was going to be used as basis for the experiments but around the start of this thesis the new YOLOv3 architecture (Redmon & Farhadi, 2018) was released. As such a comparative test was done between the two architectures to better measure the improvements made to the new algorithm. This test is presented in the following section. Given the better detection performance of the YOLOv3 algorithm, it was used as the basis for the remaining tests throughout this thesis.

4.1 COMPARISON BETWEEN YOLOV2 AND YOLOV3

In this section the results of the experiment that lead to the choice of using YOLOv3 over YOLOv2 is explained in more detail. While the inner workings of the YOLOv3 have been previously explored, there are some differences between the two algorithms that are worthy of note and that help explain the differences in detection performance. Unlike the YOLOv3, the YOLOv2 architecture has no shortcut layers, instead having max-pooling layers alternating with the convolutional layers. The YOLOv2 does not make use of the multiple paths either having only a single output layer. The architecture of YOLOv2 can be found in detail in Table 7. In order to make the intended comparison both networks were trained in darknet for 10 hours and the mAP scores of both were then compared.

As can be observed in Table 8 the YOLOv3 easily outperforms the YOLOv2 network which is within expectations. The YOLOv3 was expected to have a higher mAP than the YOLOv2 as the former is a more recent architecture created specifically with the shortcomings the previous YOLO architectures in mind. It uses modern CNN concepts not yet incorporated into the YOLOv2 to attempt to address these shortcomings and as can be seen by the results of the experiment it succeeds. Despite the apparent predictability of these results it was important to perform this experiment so as to show that the improvements made to YOLOv3 were useful in our specific problem. Interestingly after training for the same time period, the YOLOv2 algorithm went through many more iterations than the YOLOv3 network. This can be explained by the fact that the YOLOv3 architecture has many more convolutional layers than the YOLOv2 network making it slower to train. In spite of the slower iterations, the YOLOv3 still got better results than the YOLOv2 showing that it learns more per iteration than the latter.

Table 7: The YOLOv2 architecture

	Layer	Filters	Filter size	Stride	Activation
	Convolutional	32	3 x 3	1	ReLU
	Max Pooling	—	2 x 2	2	ReLU
	Convolutional	64	3 x 3	1	ReLU
	Max Pooling	—	2 x 2	2	ReLU
	Convolutional	128	3 x 3	1	ReLU
	Convolutional	64	1 x 1	1	ReLU
	Convolutional	128	3 x 3	1	ReLU
	Max Pooling	—	2 x 2	2	ReLU
	Convolutional	256	3 x 3	1	ReLU
	Convolutional	128	1 x 1	1	ReLU
	Convolutional	256	3 x 3	1	ReLU
	Max Pooling	—	2 x 2	2	ReLU
	Convolutional	512	3 x 3	1	ReLU
2x	Convolutional	256	1 x 1	1	ReLU
	Convolutional	512	3 x 3	1	ReLU
	Max Pooling	—	2 x 2	2	ReLU
	Convolutional	1024	3 x 3	1	ReLU
2x	Convolutional	512	1 x 1	1	ReLU
3x	Convolutional	1024	3 x 3	1	ReLU
	Route	—	—	—	—
	Convolutional	64	1 x 1	1	ReLU
	Route	—	—	—	—
	Convolutional	1024	3 x 3	1	ReLU
	Detection	—	—	—	—

Table 8: YOLOv2 and YOLOv3 comparison, number of iterations between parenthesis

	YOLOv2	**YOLOv3**
2 hours	37.71% (3600)	44.01% (2200)
4 hours	55.39% (7400)	60.91% (4000)
6 hours	56.27% (11100)	69.13% (5500)
8 hours	62.54% (16000)	73.41% (7500)
10 hours	63.91% (19300)	**76.06%** (9700)

4.2 REMOVAL OF LAYERS

During the following experiments, the YOLOv3 has one of its layers removed in different locations of the network to try to measure the importance of layers in specific locations inside the network. In particular, the experiments aimed to measure weather layers found deeper in the network are more or less relevant to the learning process. Three variations were tested:

- The nolayer-beginning test uses the YOLOv3 network with its 5th layer removed. This layer has 128 filters of size 3x3. It represents the relevance of early layers in the network.
- The nolayer-middle test uses the YOLOv3 network with its 67th layer removed. This layer has 1024 filters of size 3x3. It represents the relevance of layers in the middle network.
- The nolayer-middle test uses the YOLOv3 network with its 100th layer removed. This layer has 256 filters of size 3x3. It represents the relevance of layers in the middle network.

The learning process during the tests can be observed in Figure 7 and the maximum mAP achieved by each network can be found in Table 9. Unfortunately, these tests did not prove very insightful as all networks performed about the same as the yolov3 presented in the graph and table as a baseline. This possibly happens because given the big size of the networks the learning capability of the network is fairly well distributed and as such the removal of any one particular layer does not carry a particularly big impact on the network, regardless of its location. On the other hand, these results lead to the idea of using smaller networks which is further explored in the next section.

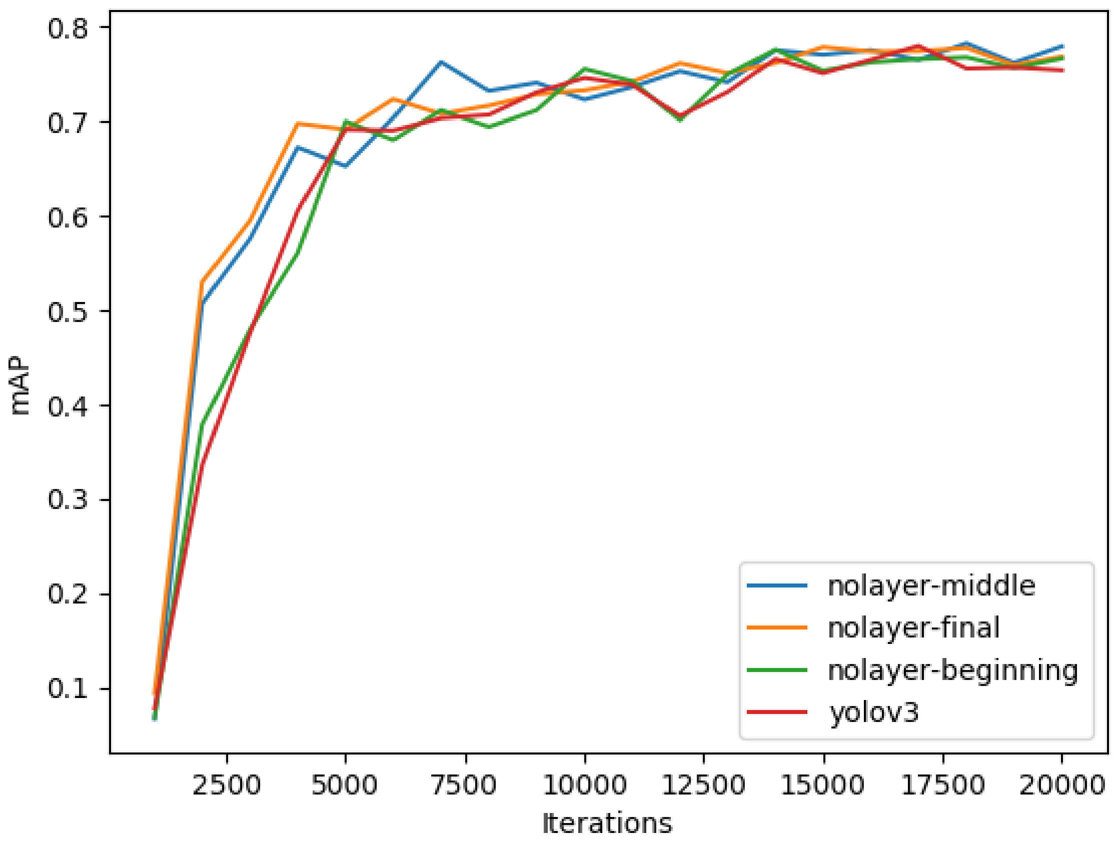

Figure 7: Graphical observation of the learning process during training for architectures with missing layers

Table 9: Individual results for testing with missing layers

yolov3	0.780366
nolayer-middle	**0.783100**
nolayer-final	0.779134
nolayer-beginning	0.776301

4.3 SMALLER SIZE NETWORK

Given the results of the experiments of the previous section, it seemed plausible that a smaller network could potentially perform just as well as a bigger network, like YOLOv3, since the problem under study is relatively less complex than general object detection for which YOLO was developed. In addition, a smaller network could make it not only faster, during training and inference, but also less computationally expensive which could be of great importance for embedded applications, as these often have restrictive computational requirements.

To test this, two architectures were created miniyolo1 and miniyolo2. The first network was developed by removing many of the repetitions of layers in the main path of the original YOLOv3. Other than this, the overall structure of the network remains the same. The miniyolo1 architecture can be seen in further detail in Table 10. This architecture has 58 layers compared to 106 of the original YOLOv3. The second network was based on miniyolo1 but had excess convolutional networks removed in the other paths as well, resulting in even fewer layers. The miniyolo2 architecture can be seen in further detail in Table 11 and has 50 layers even less the miniyolo1 and less than half of the original YOLOv3. Both of these networks are faster at inference than the YOLOv3 with miniyolo1 achieving 11 milliseconds and the miniyolo2 achieving 10 milliseconds when compared to YOLOv3 which has a 19 milliseconds inference time.

While a fast detection system is a good goal to strive towards it is meaningless if to achieve this it has severely sacrifice its accuracy. The fact that cutting the size of the model would result in quicker inference is logical but its usefulness is dependant on what kind of tradeoffs result from this. To ascertain the practicality of new models they were trained on the IVOD dataset and the learning process can be seen in Figure 8 and the final results are presented in Table 12. As we can see all three networks have similar mAP scores and these results show that the YOLOv3 network is unnecessarily large for the problem at hand. While it can be enticing to simply use an already pre-trained network from a well known and studied architecture when starting a new software project using Deep Learning, this might not be the best approach. But many of these networks are designed with general object detection in mind and as such require far more complex models than most real-world use cases require. Designing a simpler network for the specific problem at hand could be a could potentially be a better course of action depending on the problem.

Table 10: The miniyolo1 architecture

	Layer	**Filters**	**Filter size**	**Stride**	**Activation**
	Convolutional	32	3 x 3	1	ReLU
	Convolutional	64	3 x 3	2	ReLU
	Convolutional	32	1 x 1	1	ReLU
1x	Convolutional	64	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	128	3 x 3	2	ReLU
	Convolutional	64	1 x 1	1	ReLU
2x	Convolutional	128	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	256	3 x 3	2	ReLU
	Convolutional	128	1 x 1	1	ReLU
2x	Convolutional	256	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	512	3 x 3	2	ReLU
	Convolutional	256	1 x 1	1	ReLU
2x	Convolutional	512	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	1024	3 x 3	2	ReLU
	Convolutional	512	1 x 1	1	ReLU
1x	Convolutional	1024	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	512	1 x 1	1	ReLU
1x	Convolutional	1024	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	256	1 x 1	1	ReLU
3x	Convolutional	512	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	128	1 x 1	1	ReLU
3x	Convolutional	256	3 x 3	1	ReLU
	Detection	—	—	—	—
#layers	58				

Table 11: The miniyolo2 architecture

	Layer	**Filters**	**Filter size**	**Stride**	**Activation**
	Convolutional	32	3 x 3	1	ReLU
	Convolutional	64	3 x 3	2	ReLU
	Convolutional	32	1 x 1	1	ReLU
1x	Convolutional	64	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	128	3 x 3	2	ReLU
	Convolutional	64	1 x 1	1	ReLU
2x	Convolutional	128	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	256	3 x 3	2	ReLU
	Convolutional	128	1 x 1	1	ReLU
2x	Convolutional	256	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	512	3 x 3	2	ReLU
	Convolutional	256	1 x 1	1	ReLU
2x	Convolutional	512	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	1024	3 x 3	2	ReLU
	Convolutional	512	1 x 1	1	ReLU
1x	Convolutional	1024	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	512	1 x 1	1	ReLU
1x	Convolutional	1024	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	256	1 x 1	1	ReLU
1x	Convolutional	512	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	128	1 x 1	1	ReLU
1x	Convolutional	256	3 x 3	1	ReLU
	Detection	—	—	—	—
#layers	50				

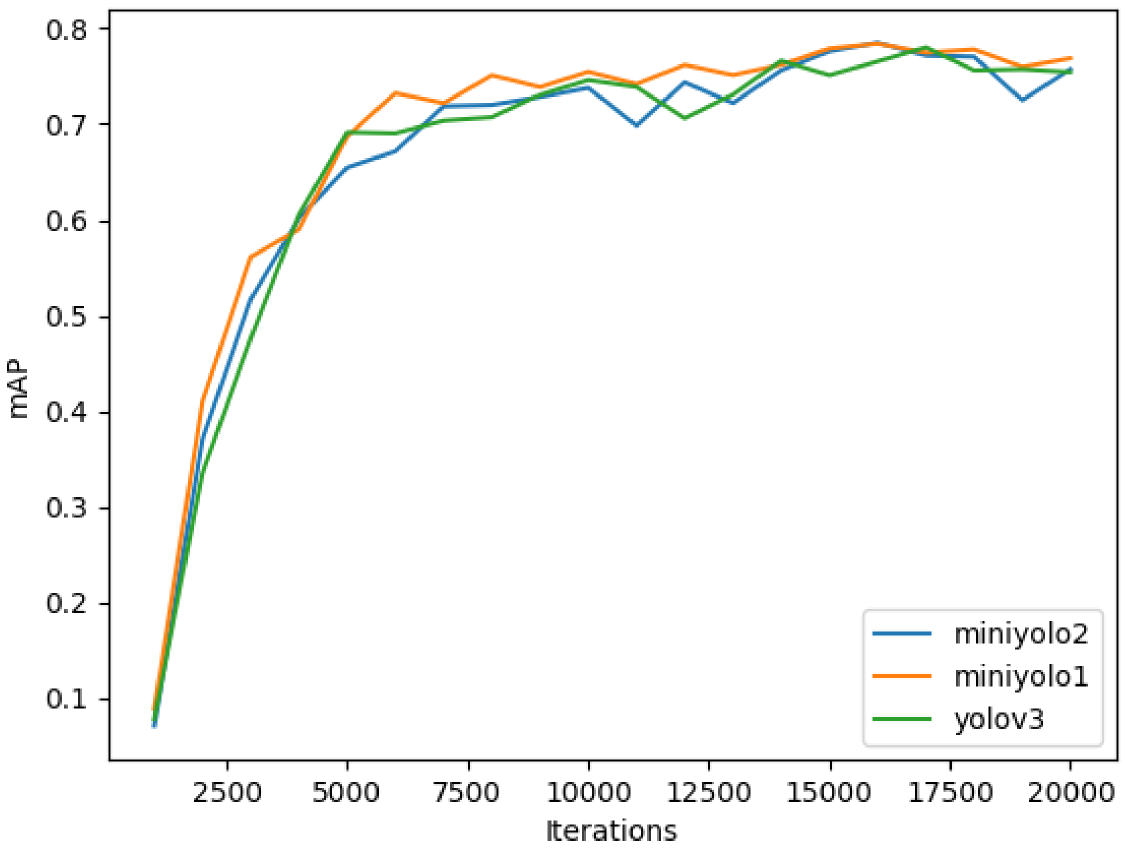

Figure 8: Graphical observation of the learning process during training for architectures with smaller networks based on YOLOv3

Table 12: Individual results for testing with smaller networks based on YOLOv3

YOLOv3	**0.785212**
miniyolo1	0.784305
miniyolo2	0.780366

In order to see how poorly the new network performed on a more general problem, the miniyolo2 was trained and tested with the smallCOCO. The results can be found in the Figure 9 and Table 13. These results came as a bit of a surprise as we expected to find that due to the increased complexity of the smallCOCO dataset the full YOLOv3 would perform much better than its smaller alternative. Instead, the two networks performed quite similarly with the miniyolo2 only losing by 1 percentage point when compared to the full YOLOv3. Given the similar size between the augmented IVOD and the smallCOCO datasets, this could indicate that the size of the dataset plays a considerably bigger role on the difficulty of a problem than what exactly is contained in the images themselves. Taking this into consideration, training data availability should become a major consideration when designing Object Detection systems.

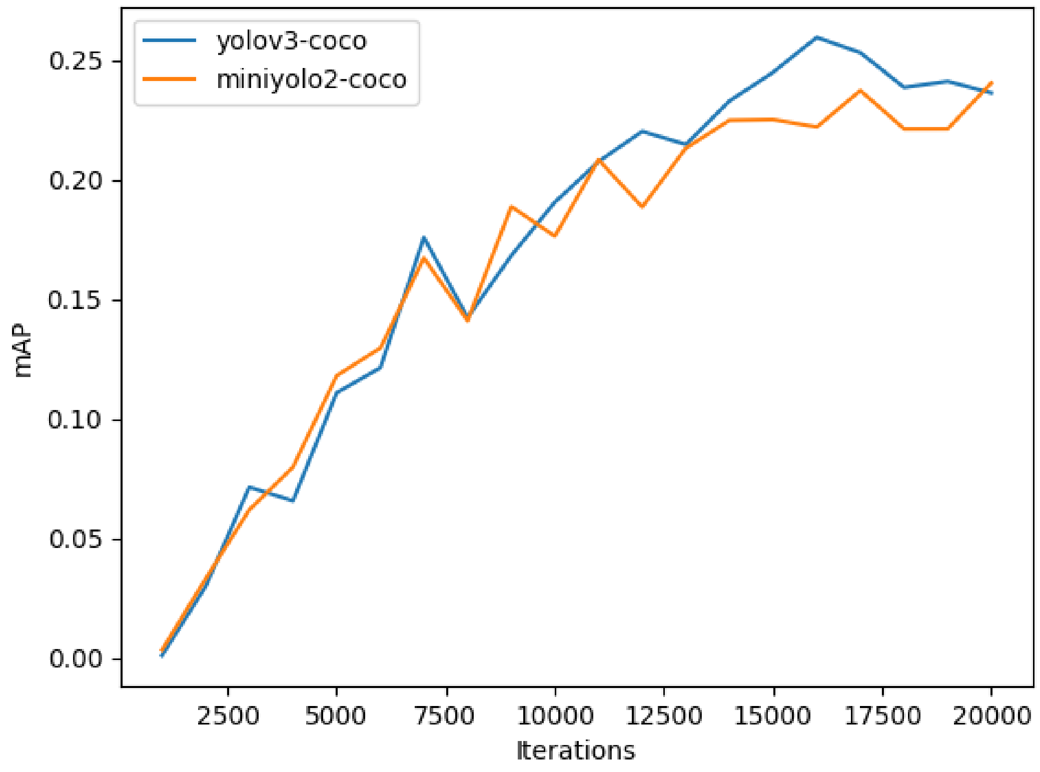

Figure 9: Graphical observation of the learning process during training with smaller networks based on YOLOv3 using smallCOCO

Table 13: Individual results for testing with smaller networks based on YOLOv3 with smallCOCO

yolov3-coco	**0.259496**
miniyolo2-coco	0.240429

Lastly, we tested the miniyolo2 on the full COCO dataset, just to be thorough, and as expected it performed much worse than the YOLOv3 indicating that for larger datasets a bigger network is, in fact, an advantage. The results from the test can be found in Figure 10 and Table 14.

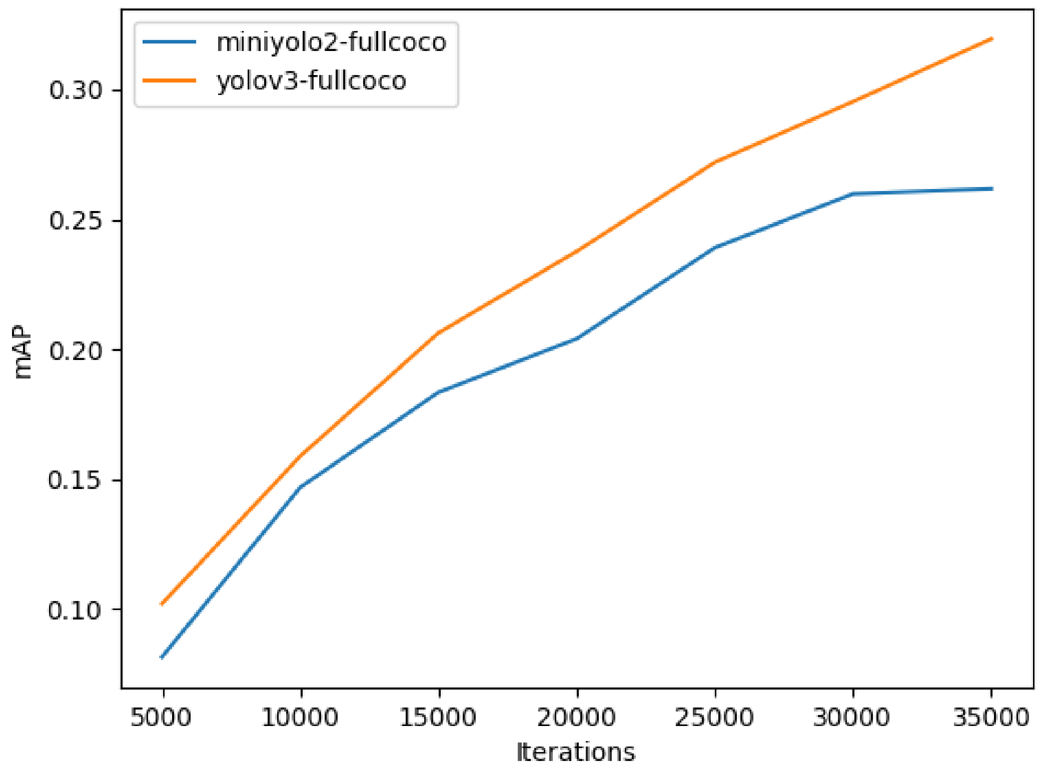

Figure 10: Graphical observation of the learning process during training with smaller networks based on YOLOv3 using COCO

Table 14: Individual results for testing with smaller networks based on YOLOv3 with COCO

yolov3-fullcoco	**0.319521**
miniyolo2-fullcoco	0.261822

4.4 FEATURE PYRAMIDS AND MULTIPLE PATHS

In order to better understand the effect that implementing Feature Pyramids has on YOLOv3 and its capabilities as an object detection algorithm, several different networks were developed around the idea of removing or adding extra paths to the original YOLOv3 architecture.

First, we created the miniyolo1-1path which starts with miniyolo1 as a basis and then removes the output layers of the first two paths. After the main path, the early high-resolution feature maps are still joined with deeper feature maps but this does not lead to different outputs. This network has only one detection layer. This architecture is presented in detail in Table 15.

Secondly, we make the miniyolo1-nopath architecture which removes the second and third path and its higher resolution feature maps altogether, from the miniyolo1 network. Just like miniyolo1-1path, this network only has one output layer. This architecture is presented in detail in Table 16.

Given that the feature pyramid paths were a new addition in YOLOv3 not present in previous versions of the algorithm we know that this feature is meant to improve the detection performance of the algorithm and indeed the original paper (Redmon & Farhadi, 2018) corroborates this. This is explained by the fact that usage of a conjunction of high-resolution and low-resolution feature maps gives the network more detailed information to work with compared to just low-resolution information as is more common in CNNs. Perhaps more surprising is the degree to which this feature improves detection performance. The miniyolo1-nopath achieves less than half the mAP of miniyolo2 which is used as a baseline. The miniyolo1-1path performs even worse which is possibly due to the fact that this network ends up with only a high-resolution output, which possibly makes it difficult to accurately learn high-level information from the images. The full results of the tests can be seen in Table 17 and Figure 11.

Table 15: The miniyolo1-1path architecture

	Layer	**Filters**	**Filter size**	**Stride**	**Activation**
	Convolutional	32	3 x 3	1	ReLU
	Convolutional	64	3 x 3	2	ReLU
	Convolutional	32	1 x 1	1	ReLU
1x	Convolutional	64	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	128	3 x 3	2	ReLU
	Convolutional	64	1 x 1	1	ReLU
2x	Convolutional	128	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	256	3 x 3	2	ReLU
	Convolutional	128	1 x 1	1	ReLU
2x	Convolutional	256	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	512	3 x 3	2	ReLU
	Convolutional	256	1 x 1	1	ReLU
2x	Convolutional	512	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	1024	3 x 3	2	ReLU
	Convolutional	512	1 x 1	1	ReLU
1x	Convolutional	1024	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	512	1 x 1	1	ReLU
1x	Convolutional	1024	3 x 3	1	ReLU
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	256	1 x 1	1	ReLU
3x	Convolutional	512	3 x 3	1	ReLU
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	128	1 x 1	1	ReLU
3x	Convolutional	256	3 x 3	1	ReLU
	Detection	—	—	—	—
#layers	50				

Table 16: The miniyolo1-nopath architecture

	Layer	Filters	Filter size	Stride	Activation
	Convolutional	32	3 x 3	1	ReLU
	Convolutional	64	3 x 3	2	ReLU
1x	Convolutional	32	1 x 1	1	ReLU
	Convolutional	64	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	128	3 x 3	2	ReLU
2x	Convolutional	64	1 x 1	1	ReLU
	Convolutional	128	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	256	3 x 3	2	ReLU
2x	Convolutional	128	1 x 1	1	ReLU
	Convolutional	256	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	512	3 x 3	2	ReLU
2x	Convolutional	256	1 x 1	1	ReLU
	Convolutional	512	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	1024	3 x 3	2	ReLU
1x	Convolutional	512	1 x 1	1	ReLU
	Convolutional	1024	3 x 3	1	ReLU
	Shortcut	—	—	—	—
1x	Convolutional	512	1 x 1	1	ReLU
	Convolutional	1024	3 x 3	1	ReLU
	Detection	—	—	—	—
#layers	34				

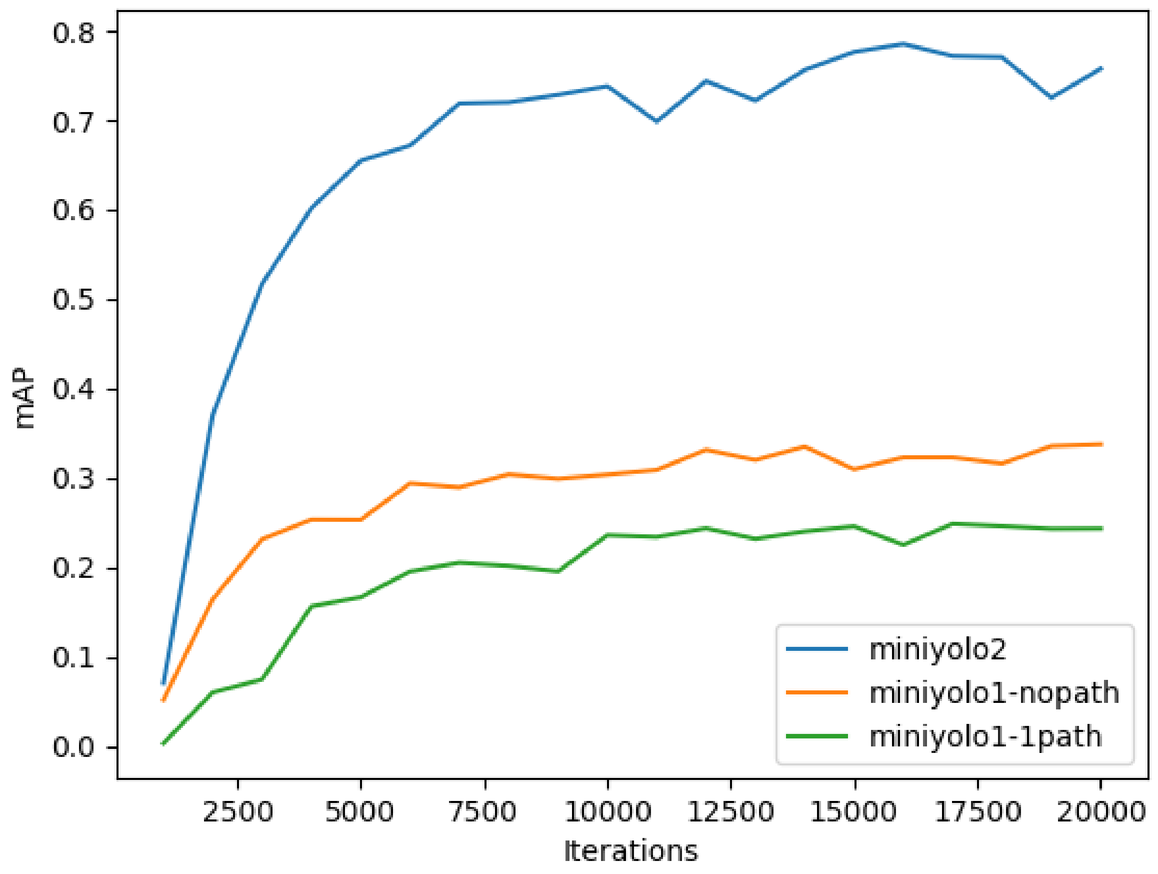

Figure 11: Graphical observation of the learning process during training with miniyolo without multiple paths

Table 17: Individual results for tests with miniYOLO without multiple paths

miniyolo2	**0.780366**
miniyolo1-nopath	0.337576
miniyolo1-1path	0.246152

We also decided to corroborate these findings in the smallCOCO dataset and the results can be found in Table 18 and Figure 12. Similarly to the previous tests, the networks without extra paths lost much of its learning capability converging early and at much lower mAP scores than the miniyolo2.

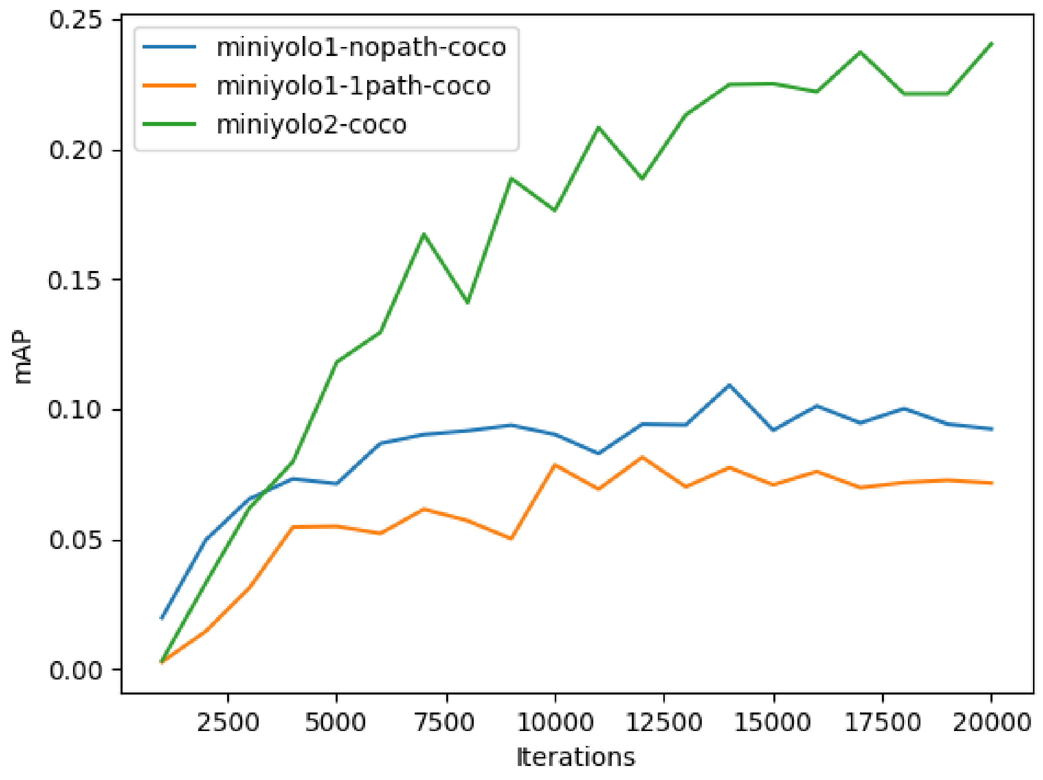

Figure 12: Graphical observation of the learning process during training with miniyolo without multiple paths with smallCOCO dataset

Table 18: Individual results for miniyolo without multiple paths tests on the smallCOCO dataset

miniyolo2-coco	**0.240429**
miniyolo1-nopath-coco	0.109271
miniyolo1-1path-coco	0.081502

Following the previous tests, we tested removing only one of the paths from the miniyolo1 architecture. The first architecture developed was the miniyolo1-12path which has the first (13x13 feature map) and second (26x26 feature map) path of the original, but the third one is missing, The second architecture has the first (13x13 feature map) and third path (52x52 feature map), but the second one is missing, and is named miniyolo1-13path. These architecture can be found in Table 19 and Table 20 respectively.

As was expected given the previous results the new networks with removed layers proved less capable than the baseline. However, the miniyolo1-12path performed better than the miniyolo1-13path. A possible explanation for this is that paths with lower resolution outputs are more useful as compared to higher resolution paths. This is interesting as the multiple paths come from the idea that as we decrease the size of the feature maps, as we go deeper into CNNs, more precise high-level information can be obtained but small-scale details of the image are often lost in the process, and as such recovering high-resolution feature maps from the shallower layers can help recover this information and improve the results obtained.

Given the results obtained in these tests, it would seem that lower resolution information is more useful and it only makes sense to introduce higher resolution paths when the lower ones have already been exhausted. The full results can be found in Table 21 and Figure 13.

Table 19: The miniyolo1-12path architecture

	Layer	**Filters**	**Filter size**	**Stride**	**Activation**
	Convolutional	32	3 x 3	1	ReLU
	Convolutional	64	3 x 3	2	ReLU
1x	Convolutional	32	1 x 1	1	ReLU
	Convolutional	64	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	128	3 x 3	2	ReLU
2x	Convolutional	64	1 x 1	1	ReLU
	Convolutional	128	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	256	3 x 3	2	ReLU
2x	Convolutional	128	1 x 1	1	ReLU
	Convolutional	256	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	512	3 x 3	2	ReLU
2x	Convolutional	256	1 x 1	1	ReLU
	Convolutional	512	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	1024	3 x 3	2	ReLU
1x	Convolutional	512	1 x 1	1	ReLU
	Convolutional	1024	3 x 3	1	ReLU
	Shortcut	—	—	—	—
1x	Convolutional	512	1 x 1	1	ReLU
	Convolutional	1024	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
3x	Convolutional	256	1 x 1	1	ReLU
	Convolutional	512	3 x 3	1	ReLU
	Detection	—	—	—	—
#layers	50				

Table 20: The miniyolo1-13path architecture

	Layer	**Filters**	**Filter size**	**Stride**	**Activation**
	Convolutional	32	3 x 3	1	ReLU
	Convolutional	64	3 x 3	2	ReLU
	Convolutional	32	1 x 1	1	ReLU
1x	Convolutional	64	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	128	3 x 3	2	ReLU
	Convolutional	64	1 x 1	1	ReLU
2x	Convolutional	128	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	256	3 x 3	2	ReLU
	Convolutional	128	1 x 1	1	ReLU
2x	Convolutional	256	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	512	3 x 3	2	ReLU
	Convolutional	256	1 x 1	1	ReLU
2x	Convolutional	512	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	1024	3 x 3	2	ReLU
	Convolutional	512	1 x 1	1	ReLU
1x	Convolutional	1024	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	512	1 x 1	1	ReLU
1x	Convolutional	1024	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	128	1 x 1	1	ReLU
3x	Convolutional	256	3 x 3	1	ReLU
	Detection	—	—	—	—
#layers	46				

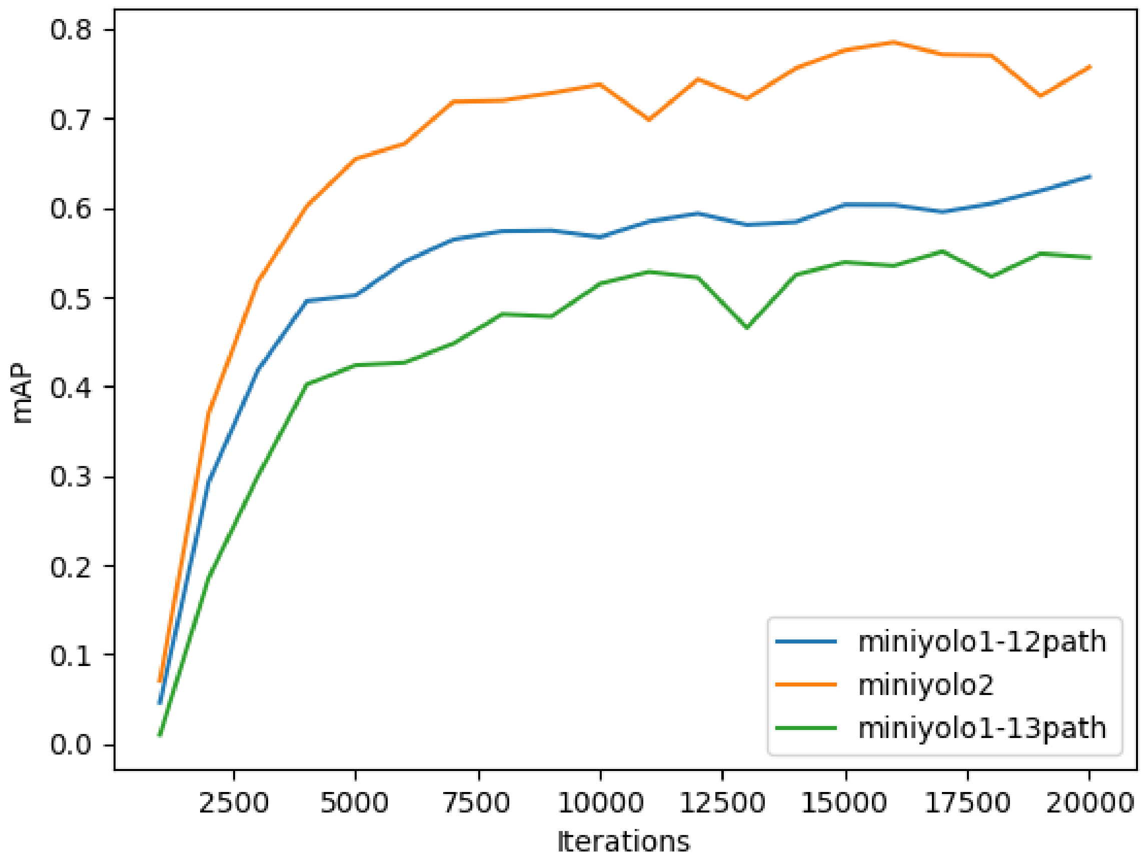

Figure 13: Graphical observation of the learning process during training with miniyolo with some paths removed

Table 21: Individual results for tests with miniyolo with some paths removed

miniyolo2	**0.780366**
miniyolo1-12path	0.634708
miniyolo1-13path	0.551173

Lastly, we tested with adding more alternate paths as this could result in further improving upon the detection performance of our networks. For these tests, we created the miniyolo2-4paths in which we added a fourth path with 104x104 feature maps and miniyolo2-5paths which built even further on the miniyolo2-4paths and added a fifth path with 208x208 feature maps.

Unfortunately, these networks achieve no better detection performance than the baseline. This indicates that, going by the assumption of the previous tests that feature maps provide less useful information the higher their resolution, feature maps above 52x52 provide no more useful information not already provided by lower resolution feature maps. The full results can be found in Table 25 and Figure 14.

Table 22: The miniyolo2-4paths architecture

	Layer	**Filters**	**Filter size**	**Stride**	**Activation**
	Convolutional	32	3 x 3	1	ReLU
	Convolutional	64	3 x 3	2	ReLU
	Convolutional	32	1 x 1	1	ReLU
1x	Convolutional	64	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	128	3 x 3	2	ReLU
	Convolutional	64	1 x 1	1	ReLU
2x	Convolutional	128	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	256	3 x 3	2	ReLU
	Convolutional	128	1 x 1	1	ReLU
2x	Convolutional	256	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	512	3 x 3	2	ReLU
	Convolutional	256	1 x 1	1	ReLU
2x	Convolutional	512	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	1024	3 x 3	2	ReLU
	Convolutional	512	1 x 1	1	ReLU
1x	Convolutional	1024	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	512	1 x 1	1	ReLU
1x	Convolutional	1024	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	256	1 x 1	1	ReLU
1x	Convolutional	512	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	128	1 x 1	1	ReLU
1x	Convolutional	256	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	64	1 x 1	1	ReLU
1x	Convolutional	128	3 x 3	1	ReLU
	Detection	—	—	—	—
#layers	58				

Table 23: The miniyolo2-5paths architecture

	Layer	Filters	Filter size	Stride	Activation
	Convolutional	32	3 x 3	1	ReLU
	Convolutional	64	3 x 3	2	ReLU
	Convolutional	32	1 x 1	1	ReLU
1x	Convolutional	64	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	128	3 x 3	2	ReLU
	Convolutional	64	1 x 1	1	ReLU
2x	Convolutional	128	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	256	3 x 3	2	ReLU
	Convolutional	128	1 x 1	1	ReLU
2x	Convolutional	256	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	512	3 x 3	2	ReLU
	Convolutional	256	1 x 1	1	ReLU
2x	Convolutional	512	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	1024	3 x 3	2	ReLU
	Convolutional	512	1 x 1	1	ReLU
1x	Convolutional	1024	3 x 3	1	ReLU
	Shortcut	—	—	—	—
	Convolutional	512	1 x 1	1	ReLU
1x	Convolutional	1024	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	256	1 x 1	1	ReLU
1x	Convolutional	512	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	128	1 x 1	1	ReLU
1x	Convolutional	256	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	64	1 x 1	1	ReLU
1x	Convolutional	128	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—

Table 24: The miniyolo2-5paths architecture (continuation)

	Layer	Filters	Filter size	Stride	Activation
	Route	—	—	—	—
	Convolutional	32	1 x 1	1	ReLU
1x	Convolutional	64	3 x 3	1	ReLU
	Detection	—	—	—	—
#layers	66				

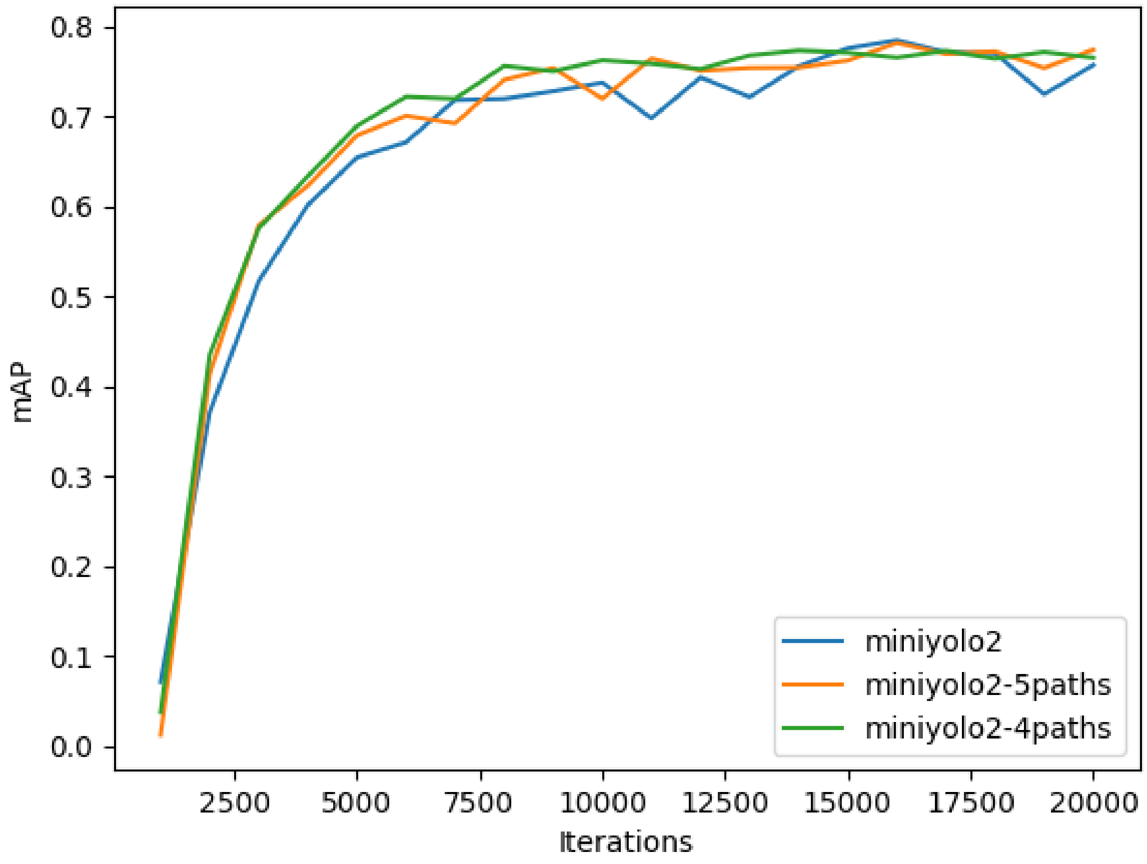

Figure 14: Graphical observation of the learning process during training with miniyolo with extra paths added

Table 25: Individual results tests with miniyolo with extra paths added

miniyolo2	**0.780366**
miniyolo2-4paths	0.773788
miniyolo2-5paths	0.780168

4.5 PARALLEL NETWORKS

In this section, we start by building an even more streamlined network without the shortcut layers. This architecture which can be found in Table 26 is called newDarkNetwork. Given the further reduction in the size of the network and the missing shortcut layers, a drop in detection performance would be expected as the usage of shortcut layers is known to increase detection performance in networks (He et al., 2016). Despite this, the new network performs about as well as the miniyolo2 which shows that the YOLO algorithm was not making much use out of its shortcut layers at least in our task of in-vehicle detection.

A second network was also developed based on the newDarkNetwork that once again cut in the number of convolutional layers. The new network can be seen in further detail in Table 27 Unlike its predecessor this new architecture performed worse than miniyolo2. Showing that this new network presents a sufficiently small number of convolutional layers that it is no longer capable of learning the IVOD dataset to degree that previous networks including YOLOv3 could. The results of the experiments with these networks can be seen in Table 28 and Figure 15.

Table 26: The newDarkNetwork architecture

	Layer	Filters	Filter size	Stride	Activation
	Convolutional	32	3 x 3	1	ReLU
	Convolutional	64	3 x 3	2	ReLU
	Convolutional	64	3 x 3	1	ReLU
	Convolutional	128	3 x 3	2	ReLU
	Convolutional	128	3 x 3	1	ReLU
	Convolutional	256	3 x 3	2	ReLU
	Convolutional	256	3 x 3	1	ReLU
	Convolutional	512	3 x 3	2	ReLU
	Convolutional	512	3 x 3	1	ReLU
	Convolutional	1024	3 x 3	2	ReLU
	Convolutional	1024	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	512	1 x 1	1	ReLU
	Convolutional	256	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	256	1 x 1	1	ReLU
	Convolutional	128	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	128	1 x 1	1	ReLU
	Convolutional	64	3 x 3	1	ReLU
	Detection	—	—	—	—
#layers	37				

Table 27: The miniDarkNetwork architecture

	Layer	**Filters**	**Filter size**	**Stride**	**Activation**
	Convolutional	32	3 x 3	1	ReLU
	Convolutional	64	3 x 3	2	ReLU
	Convolutional	128	3 x 3	2	ReLU
	Convolutional	256	3 x 3	2	ReLU
	Convolutional	512	3 x 3	2	ReLU
	Convolutional	1024	3 x 3	2	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	512	1 x 1	1	ReLU
	Convolutional	256	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	256	1 x 1	1	ReLU
	Convolutional	128	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	128	1 x 1	1	ReLU
	Convolutional	64	3 x 3	1	ReLU
	Detection	—	—	—	—
#layers	24				

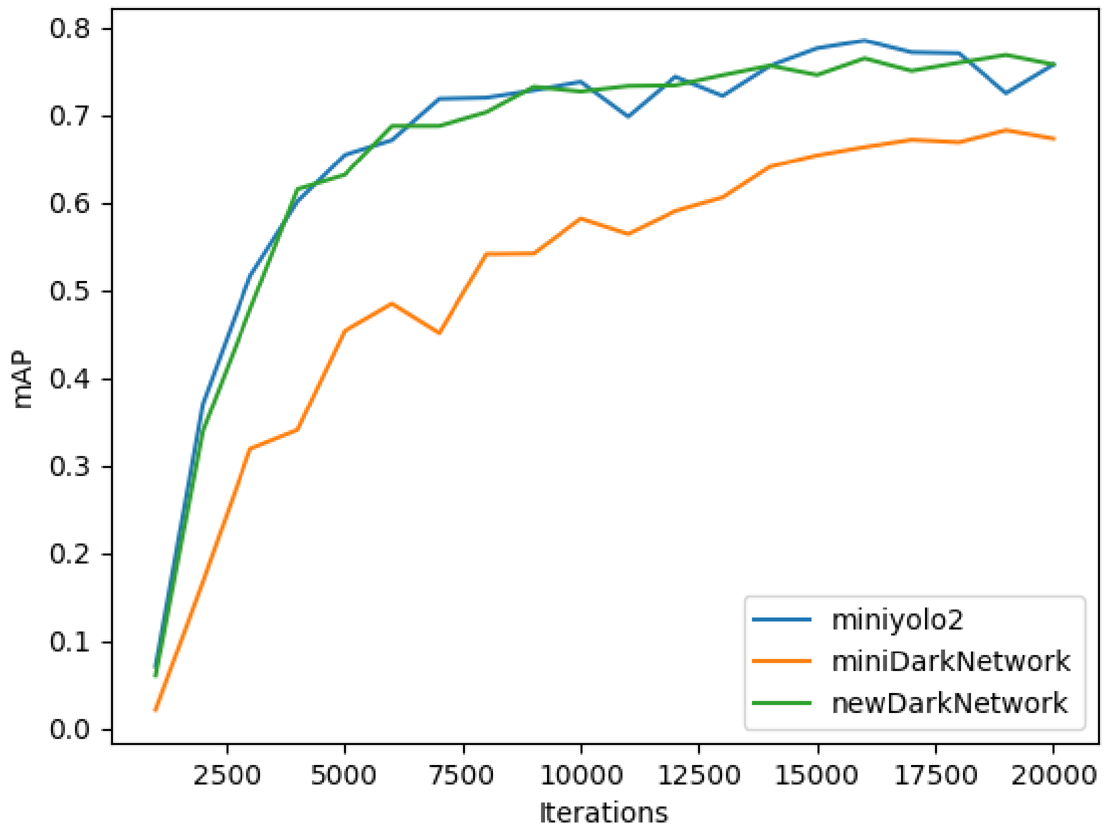

Figure 15: Graphical observation of the learning process during training with new architectures developed without shortcut layers

Table 28: Individual results for tests with new architectures developed without shortcut layers

miniyolo2	**0.785212**
newDarkNetwork	0.768622
miniDarkNetwork	0.683027

Finally, the miniDarkNetwork architecture developed in the previous tests is used to test whether the learning algorithms can make use of separate parallel networks and make them each learn separate information while being trained as a whole network. The first architecture developed is basically two copies of the miniDarkNetwork both receiving input from the same initial layer. The two copies have no connection and both have their own output layers. This architecture is called parallelMiniDark and is presented in detail in Table 29.

The second architecture developed is similar to the parallelMiniDark except that the outputs of both the miniDarkNetwork feed their outputs to a convolutional layer which then is fed into a new output layer. This second network aims to find out how fusing the results of the parallel networks affects the overall detection performance of the whole network. It is called parallelMiniDarkFusion and its full architecture can be found in Table 30.

Both of these network performed about as well as the original miniDarkNetwork from which both the parallel networks were built. This means that these new networks have twice the amount of convolutional layers but they perform about the same as a single one which shows that the learning algorithm is incapable of making use of these layers distributed in parallel to learn further information. This ends up being a waste of resources and not worthwhile. Lastly, it would seem that the fusion made no difference in addressing this problem of the training system not being able to make full use of parallel networks. The results of the experiments with these networks can be seen in Table 31 and Figure 16.

Table 29: The parallelMiniDark architecture

	Layer	Filters	Filter size	Stride	Activation
	Convolutional	32	3 x 3	1	ReLU
	Convolutional	64	3 x 3	2	ReLU
	Convolutional	128	3 x 3	2	ReLU
	Convolutional	256	3 x 3	2	ReLU
	Convolutional	512	3 x 3	2	ReLU
	Convolutional	1024	3 x 3	2	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	512	1 x 1	1	ReLU
	Convolutional	256	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	256	1 x 1	1	ReLU
	Convolutional	128	3 x 3	1	ReLU
	Detection	—	—	—	—
	Route 0	—	—	—	—
	Convolutional	32	3 x 3	1	ReLU
	Convolutional	64	3 x 3	2	ReLU
	Convolutional	128	3 x 3	2	ReLU
	Convolutional	256	3 x 3	2	ReLU
	Convolutional	512	3 x 3	2	ReLU
	Convolutional	1024	3 x 3	2	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	512	1 x 1	1	ReLU
	Convolutional	256	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	256	1 x 1	1	ReLU
	Convolutional	128	3 x 3	1	ReLU
	Detection	—	—	—	—
#layers	48				

Table 30: The parallelMiniDarkFusion architecture

	Layer	Filters	Filter size	Stride	Activation
	Convolutional	32	3 x 3	1	ReLU
	Convolutional	64	3 x 3	2	ReLU
	Convolutional	128	3 x 3	2	ReLU
	Convolutional	256	3 x 3	2	ReLU
	Convolutional	512	3 x 3	2	ReLU
	Convolutional	1024	3 x 3	2	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	512	1 x 1	1	ReLU
	Convolutional	256	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	256	1 x 1	1	ReLU
exit1	Convolutional	128	3 x 3	1	ReLU
	Detection	—	—	—	—
	Route 0	—	—	—	—
	Convolutional	32	3 x 3	1	ReLU
	Convolutional	64	3 x 3	2	ReLU
	Convolutional	128	3 x 3	2	ReLU
	Convolutional	256	3 x 3	2	ReLU
	Convolutional	512	3 x 3	2	ReLU
	Convolutional	1024	3 x 3	2	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	512	1 x 1	1	ReLU
	Convolutional	256	3 x 3	1	ReLU
	Detection	—	—	—	—
	Upsample 2x	—	—	—	—
	Route	—	—	—	—
	Convolutional	256	1 x 1	1	ReLU
	Convolutional	128	3 x 3	1	ReLU
exit2	Detection	—	—	—	—
	Route exit1 + exit2	—	—	—	—
	Convolutional	128	3 x 3	1	ReLU
	Detection	—	—	—	—
#layers	52				

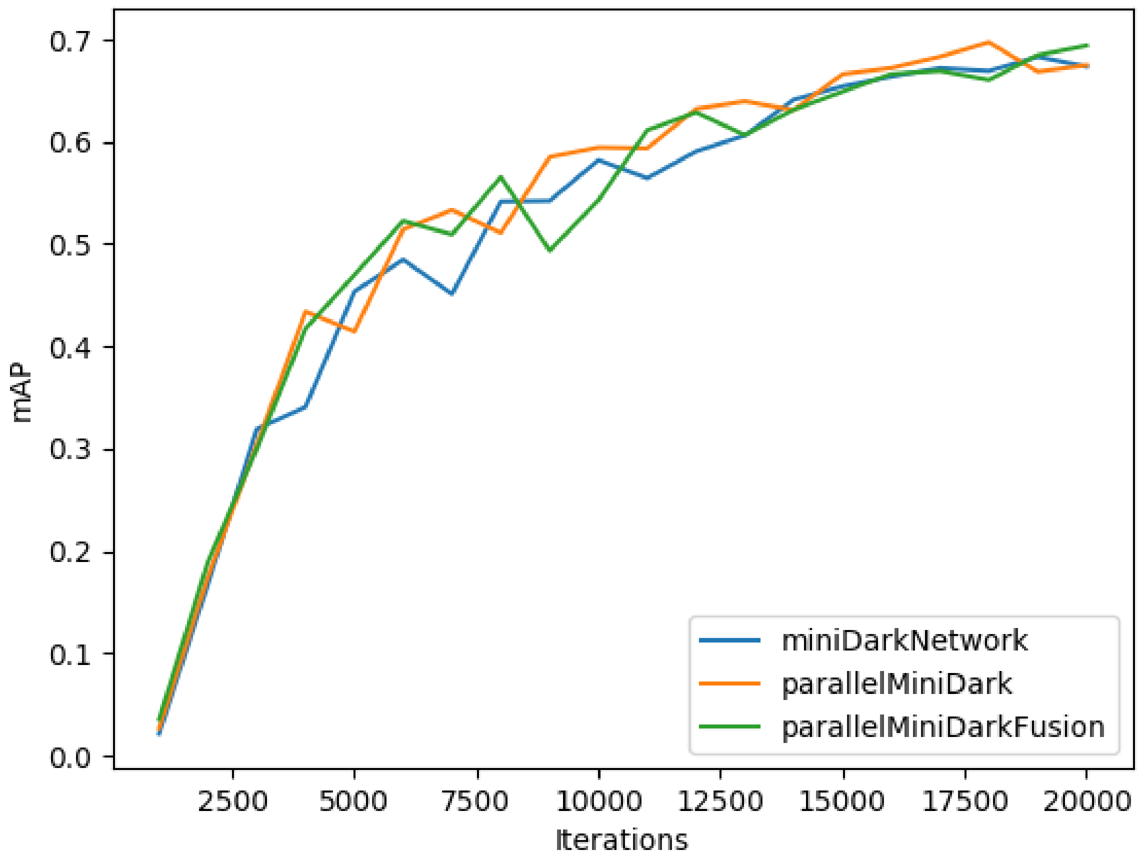

Figure 16: Graphical observation of the learning process during training with parallel networks

Table 31: Individual results for parallel networks tests

miniDarkNetwork	0.683027
parallelMiniDark	**0.697324**
parallelMiniDarkFusion	0.694057

4.6 LEARNING RATE AND BATCH SIZE

Finally, we made some tests to try to discover if the default values of batch size and learning rate of YOLOv3 are still the best ones for the task of in-vehicle object detection with our new miniyolo2 network. The batch size and learning rate are two hyper-parameters that are common with ANNs. The batch size specifies how many images are given to the algorithm per iteration. On the one hand, bigger batches allow the algorithm to make better use of parallelism making better use of the available hardware, it also makes it easier to converge as it is less likely for some batches to be either much easier or much harder to learn than average. On the other hand, small batch sizes have been associated with regularization effects ANNs, increasing the generalization ability of the algorithms (Goodfellow et al., 2016).

Learning rate is a number that indicates how big a learning step the algorithm should take each iteration. When it is too small the learning algorithm takes too long to converge as each weight update is very small. However, if the learning rate is too big the algorithm could fail to generalize altogether by becoming stuck in local minima, causing instability (Goodfellow et al., 2016).

The default values for batch size and learning rate are 64 and 0.001 respectively for the YOLOv3 algorithm. In the tests, a total of 5 other possible values were attempted. For batch sizes, we tried 128, 32 and 16. For learning rate, we tried 0.01 and 0.1. The results of these tests can be found in Figure 17 and in Table 32. Of the values tested only changing the learning rate from the original 0.001 to 0.01 resulted in any kind of improvements. Interestingly the Darknet framework seems to be incapable of learning with sufficiently high learning rates. As can be seen in Figure 17 for learning 0.1 the algorithm does not learn at all, possibly because this causes some kind of overflow during the training.

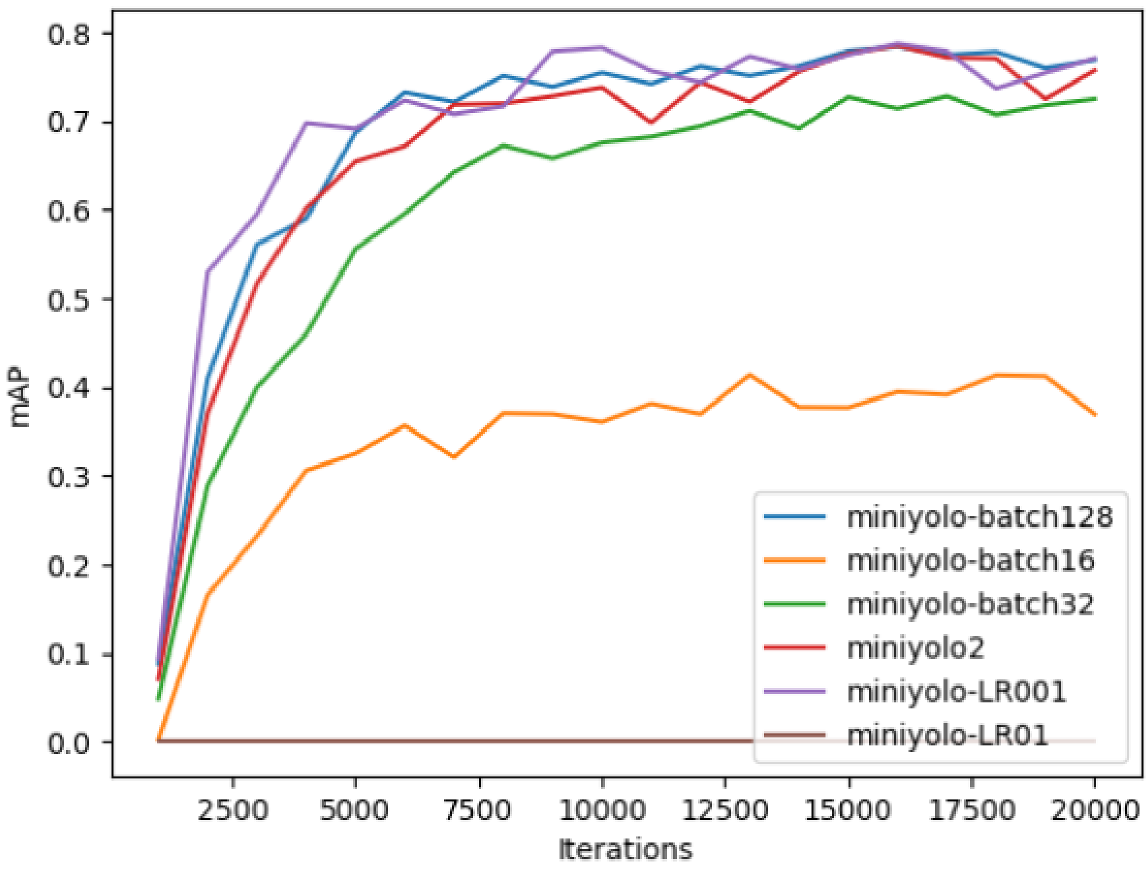

Figure 17: Graphical observation of the learning process during training for learning rate and batch size testing

Table 32: Individual results for learning rate and batch size tests

miniyolo	0.785212
miniyolo-batch128	0.784305
miniyolo-batch32	0.727432
miniyolo-batch16	0.414269
miniyolo-LR01	0.0
miniyolo-LR001	**0.787652**

5

CONCLUSIONS AND FUTURE WORK

Throughout this thesis, the YOLOv3 network was evaluated in a variety of circumstances on the problem of in-vehicle object detection. To this end a dataset was collected for the purpose of this research, that aims to represent the wide range of possibilities inherent to this kind of problem. Given the detection performance of YOLO, a quick and accurate object detection deep learning system, several experiments were made to glean further insight into the inner workings of the YOLO algorithm how it can be better applied to the problem under study.

A comparative analysis was made between the YOLOv3 and its predecessor the YOLOv2 was made to better understand how the new algorithm can be used to increase the accuracy of an object detection system for in-vehicle use. Of particular note are the addition of multiple detection paths with different feature map resolutions and how much deeper the network in itself is. Despite the apparent usefulness of these advances in comparison to YOLOv2 their usefulness for in-vehicle detection is put into question by other tests.

A new architecture miniyolo2 is developed specifically for the problem and despite its much smaller size than the full YOLOv3 it still manages to achieve the same accuracy. This not only could be very useful for real-world applications given the limitations of embedded systems but also puts into question the apparent wisdom of using a pre-prepared and well-studied architecture that is developed for more general problems as is the case with most of the famous CNN architectures used in Computer Vision problems. For real-world applications, particularly in setups with heavy hardware constraints, the development of entirely new purpose-built architectures should be considered.

A deeper study of the effect of multiple paths of different resolutions in YOLOv3 was done not only to evaluate its importance for the architecture but how it could potentially be used to improve other architectures and how to best implement it. The usage of Feature Pyramids shows to be a highly profitable endeavor as it proves responsible for much of the improvement in YOLOv3. This being said this proposal relies on the idea that each successive path added will have higher resolution than their predecessors but that lower resolutions are more useful to the network. As one would expect this technique offers diminishing returns and at least for YOLO in our dataset it does not seem to be worthwhile to add more than 2 extra paths already present int he original YOLOv3.

A new architecture was also developed to test how well deep learning systems could potentially make use of parallel, rather than sequential, layers. Given the usefulness of the Inception module (Szegedy et al., 2015) that uses this idea but on a more local scale with only a few layers each time. This route of study proved unsuccessful however as the tests did not show any great improvement. Worthy of note is the fact that unlike with the usage of multiple paths having one output with the results from both parallel networks and having multiple outputs did not make much difference.

Lastly, an exploratory study was made in regards to the default values for the learning rate and batch size of the YOLOv3 network. It was found that in regards to our problem increasing the learning rate could increase the results of the system but that if increased too much the learning rate could result in the complete inability of the network to learn.

From all these tests we end up presenting a well rounded demonstration of not only the potential of the YOLOv3 architecture for a variety problems but also how modifying it to fit the problem at hand can bring great advantages and how it can possibly be used in the future for in-vehicle object detection as this is no doubt an area of great interest with the increasing adherence of the car industry to the usage of intelligent systems to improve vehicle usability.

Despite the results obtained in this thesis the usage of deep learning in in-vehicle object detection is still very much an open problem as is object detection in general, and as such this section presents not only other paths of possible study that could not be explored in this thesis but also further experimentation that could be done in the paths that were explored.

For starters, the IVOD dataset used in the experiments performed has several issues that are relevant to acknowledge as part of the context of this thesis. This dataset is rather small for the dimension of the problem. At a little over 3000 images, the IVOD is much smaller than the datasets usually used in the state of the art such as the COCO detection dataset with around 500000 images. This has the advantage of making training and testing much quicker which gives the possibility to perform more tests within the same time frame. Now while this smaller size is to be expected to a degree, as the COCO dataset is far more general in scope including 80 different types of objects in any context, this is still a relatively small dataset. And as such real-world data in greater quantity could greatly increase our understanding of this subject.

While the results of the exploration into parallel networks proved unfruitful further research into this idea could still prove interesting particularly if the parallel networks used are slightly different introducing some asymmetry.

REFERENCES

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2014). Semantic image segmentation with deep convolutional nets and fully connected crfs. *CoRR*, *abs/1412.7062*.

Coco detection leaderboard. (n.d.). Retrieved 2018-09-10, from `http://cocodataset.org/#detection-leaderboard` ([ACESSED] 2018-09-10)

Darkflow github. (n.d.). Retrieved 2018-10-09, from `https://github.com/thtrieu/darkflow` ([ACESSED] 2018-10-09)

Dechter, R. (1986). Learning while searching in constraint-satisfaction-problems. In *Proceedings of the 5th national conference on artificial intelligence. philadelphia, pa, august 11-15, 1986. volume 1: Science.* (pp. 178–185). Retrieved from `http://www.aaai.org/Library/AAAI/1986/aaai86-029.php`

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & fei Li, F. (2009). Imagenet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248-255.

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, *55*(10), 78–87.

Erhan, D., Bengio, Y., Courville, A. C., Manzagol, P.-A., Vincent, P., & Bengio, S. (2010). Why does unsupervised pre-training help deep learning? In *Journal of machine learning research.*

Girshick, R. B. (2015). Fast r-cnn. *2015 IEEE International Conference on Computer Vision (ICCV)*, 1440-1448.

Girshick, R. B., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 580-587.

Goodfellow, Bengio, & Courville. (2016). *Deep learning*. MIT Press. (`http://www.deeplearningbook.org`)

Goodfellow, Bengio, Courville, & Hinton. (2015). Deep learning. *Nature*, *521 7553*, 436-44.

Han, X., Zhong, Y., Cao, L., & Zhang, L. (2017). Pre-trained alexnet architecture with pyramid pooling and supervision for high spatial resolution remote sensing image scene classification. *Remote Sensing*, *9*, 848.

Handbook of computer vision and applications with cdrom. (1999). In B. Jahne, P. Geissler, & H. Haussecker (Eds.), (1st ed., Vol. 3, p. 4-6). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770-778.

Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, *18 7*, 1527-54.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, *abs/1412.6980*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Commun. ACM*, *60*, 84-90.

Large scale visual recognition challenge 2012 (ilsvrc2012) results. (n.d.). Retrieved 2018-09-10, from http://www.image-net.org/challenges/LSVRC/2012/results.html ([ACESSED] 2018-09-10)

Learned-Miller, E. G. (2011). Introduction to computer vision..

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, *1*(4), 541–551.

Lin, T., Dollár, P., Girshick, R. B., He, K., Hariharan, B., & Belongie, S. J. (2016). Feature pyramid networks for object detection. *CoRR*, *abs/1612.03144*. Retrieved from http://arxiv.org/abs/1612.03144

Lin, T., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., ... Zitnick, C. L. (2014). Microsoft COCO: common objects in context. *CoRR*, *abs/1405.0312*. Retrieved from http://arxiv.org/abs/1405.0312

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Iccv.*

Minsky, M., & Papert, S. (1969). *Perceptrons: An introduction to computational geometry*. MIT Press.

Mitchell, T. M. (1997). *Machine learning* (1st ed.). New York, NY, USA: McGraw-Hill, Inc.

Murphy-Chutorian, E., & Trivedi, M. M. (2009). Head pose estimation in computer vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *31*, 607-626.

Redmon, J. (2013–2016). *Darknet: Open source neural networks in c.* http://pjreddie.com/darknet/.

Redmon, J., Divvala, S. K., Girshick, R. B., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 779-788.

Redmon, J., & Farhadi, A. (2017). Yolo9000: Better, faster, stronger. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6517-6525.

Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. *CoRR*, *abs/1804.02767*.

Ren, S., He, K., Girshick, R. B., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*, 1137-1149.

Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton project para*. Cornell Aeronautical Laboratory.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. In D. E. Rumelhart, J. L. McClelland, & C. PDP Research Group (Eds.), (pp. 318–362). Cambridge, MA, USA: MIT Press. Retrieved from http://dl.acm.org/citation.cfm?id=104279.104293

Sebe, N., Cohen, I., Garg, A., & Huang, T. S. (2005). Machine learning in computer vision. In *Computational imaging and vision.*

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., & LeCun, Y. (2013). Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, *abs/1312.6229*.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, *abs/1409.1556*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., ... Rabinovich, A. (2015). Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1-9.

Szegedy, C., Reed, S. E., Erhan, D., & Anguelov, D. (2014). Scalable, high-quality object detection. *CoRR*, *abs/1412.1441*.

Szeliski, R. (2010). Computer vision: Algorithms and applications. In (1st ed., p. 11-18). New York, NY, USA: Springer-Verlag New York, Inc.

Uijlings, J. R. R., van de Sande, K. E. A., Gevers, T., & Smeulders, A. W. M. (2013). Selective search for object recognition. *International Journal of Computer Vision*, *104*(2), 154–171. Retrieved from https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013

Yilmaz, A., Javed, O., & Shah, M. (2006). Object tracking: A survey. *ACM Comput. Surv.*, *38*, 13.

Yu, D., Deng, L., & Dahl, G. E. (2010). Roles of pre-training and fine-tuning in context-dependent dbn-hmms for real-world speech recognition..

Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Eccv.*

This thesis was written under a curricular internship at **Bosch Car Multimedia Portugal S.A.**