NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

BSc THESIS

Deep Learning Techniques on Object Detection with Application on Vehicle Detection

Vasileios P. Sakkas

Supervisor: **Dimitrios Gunopoulos,** Professor

ATHENS

JULY 2019

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Τεχνικές Βαθιάς Μάθησης στον Εντοπισμό Αντικειμένων με Εφαρμογή στον Εντοπισμό Οχημάτων

Βασίλειος Π. Σακκάς

Επιβλέπων: **Δημήτριος Γουνόπουλος,** Καθηγητής

ΑΘΗΝΑ

ΙΟΥΛΙΟΣ 2019

BSc THESIS

Deep Learning Techniques on Object Detection with Application on Vehicle Detection

Vasileios P. Sakkas
S.N.: 1115201400175

SUPERVISOR: **Dimitrios Gunopoulos,** Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Τεχνικές Βαθιάς Μάθησης στον Εντοπισμό Αντικειμένων με Εφαρμογή στον Εντοπισμό Οχημάτων

Βασίλειος Π. Σακκάς

Α.Μ.: 1115201400175

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Δημήτριος Γουνόπουλος,** Καθηγητής

ABSTRACT

In recent years, there have been major improvements in computer vision tasks such as image classification and object detection, both in accuracy and performance. This improvement can be attributed to two factors, large labeled image datasets such as ImageNet that have been available for the last few years, and the rise of convolutional neural networks, which can benefit from these datasets, that have a near human level performance in these tasks.

In this Thesis, we will implement and use VGGNet, a deep Convolutional Neural Network that was submitted to the ImageNet Challenge 2014. More specifically, we will use two of its variants, VGG16 and VGG19 in order to detect and provide bounding boxes for vehicles found in CCTV footage. For this task, we will go through the training process of the two Neural Networks, compare two different methods for providing Region Proposals to the networks, Sliding Windows and Selective Search, and use two different ways to see how well our object detection system works, Accuracy and mean Average Precision.

SUBJECT AREA: Deep Learning

KEYWORDS: VGGNet, Object Detection, Convolutional Neural Networks, Selective Search, Sliding Windows, Image processing

ΠΕΡΙΛΗΨΗ

Τα τελευταία χρόνια, έχουν σημειωθεί σημαντικές βελτιώσεις σε διάφορους τομείς της μηχανικής όρασης, όπως η αναγνώριση και ο εντοπισμός αντικειμένων, τόσο σε ακρίβεια όσο και σε απόδοση. Αυτή η βελτίωση μπορεί να αποδοθεί σε δύο παράγοντες, μεγάλα κατηγοριοποιημένα σύνολα εικόνων όπως το ImageNet που είναι διαθέσιμα τα τελευταία χρόνια, και η άνοδος των Συνελικτικών Νευρωνικών Δικτύων που μπορούν να επωφεληθούν από αυτά τα μεγάλα σύνολα δεδομένων και που έχουν ακρίβεια συγκρίσιμη με αυτήν ανθρώπων σε προβλήματα μηχανικής όρασης.

Στην παρούσα εργασία θα υλοποιήσουμε το VGGNet, ένα βαθύ Συνελικτικό Νευρωνικό Δίκτυο που υποβλήθηκε στο ImageNet Challenge 2014. Ειδικότερα, θα χρησιμοποιήσουμε δύο από τις μορφές του, VGG16 και VGG19, για να ανιχνεύσουμε και να παράγουμε πλαίσια οριοθέτησης για οχήματα που βρίσκονται σε εικόνες από CCTV υλικό. Θα περιγράψουμε την διαδικασία εκπαίδευσης των δύο Νευρωνικών Δικτύων, θα συγκρίνουμε δύο διαφορετικές μεθόδους για την παροχή πλαισίων στα δύο δίκτυα, τα Sliding Windows και την Selective Search και θα χρησιμοποιήσουμε δύο διαφορετικούς τρόπους για να δούμε πόσο καλά λειτουργεί το σύστημα ανίχνευσης αντικειμένων μας, ακρίβεια (Accuracy) και mean Average Precision.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Βαθιά Μάθηση

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: VGG-Net, Ανίχνευση Αντικειμένων, Συνελικτικά Νευρωνικά Δίκτυα, Επιλεκτική Αναζήτηση, Συρόμενα Παράθυρα, Επεξεργασία Εικόνας

To my family, for their support, understanding and love throughout these years.

AKNOWLEDGMENTS

For the present Bachelor's Thesis, I would like to thank my supervisor, professor Dimitrios Gunopulos, for the guidance and the advice that contributed towards successfully completing the work.

CONTENTS

LIST OF FIGURES

LIST OF IMAGES

LIST OF TABLES

PREFACE

The thesis at hand was undertaken as part of the course of study for the undergraduate degree at the Department of Informatics and Telecommunications of the National and Kapodistrian University of Athens. The relevant work was conducted from April 2018 to July 2019 in Athens under the supervision of professor Dimitrios Gunopulos.

I would like to thank the supervising professor for the guidance he provided over the course of the project. In addition, I would like to thank the members of the KDD lab for providing the dataset that was used for the training and testing.

1. INTRODUCTION

1.1 Background

Computer vision has been an interesting topic where many improvements have been made in the last few years. Machine Learning methods and more recently, Deep Learning models have been used for image classification and object detection, with the first one meaning that we're retrieving a label describing an object and the later one meaning that we're finding one or more bounding boxes of objects inside one image, while even image segmentation, which means assigning parts of an image on a pixel per pixel level to objects has become possible.

Part of these improvements can be attributed to two factors. The first one is the appearance of large training datasets such as ImageNet[2], which contains images in the range of several millions of images which is significantly higher compared to the size of other well known image datasets, such as Caltech-101/256[4,5] and CIFAR-10/100[6] which are in the range of tens of thousands.

The second factor is the vast improvements that have been made through various Deep Learning models, and more specifically Convolutional Neural Networks. Having access to datasets with millions of labeled images of real-world settings and objects has allowed to design and successfully train models of increased depth and complexity which can perform better compared to other methods on complex computer vision tasks, where their accuracy and performance can even be compared to that of humans.

1.2 Related Work

As mentioned above, the ImageNet[2] dataset has played an important role in computer vision since through that, many new Deep Learning models have been designed, tested and distributed. More specifically, the ImageNet Large Scale Visual Recognition Challenege (ILSVRC)[3] is an annual contest where software programs compete in image classification and object detection tasks. For the classification task, submitted solutions are used to classify images from a total of 1000 classes with one of the metrics used to rank the programs being the top-5 error rate, meaning that if one of the top 5 predictions per image, or in other words, one of the 5 predicted labels with the highest probability, is the correct one, then that specific prediction is considered correct.

The first Deep Learning model that significantly outperformed all other solutions on ILSVRC[3] was AlexNet[7], a deep Convolutional Neural Network that got the first place in 2012. It was one of the first models to take advantage of large training datasets and the high performance of modern GPUs and it managed to reduce the top-5 error rate from 25.2%, which was the lowest error rate from 2011, to just 15.3%. It consists of 8 layers, 5 convolutional and 3 fully connected and it utilized several techniques in order to improve the results and reduce overfitting such as using ReLU, Softmax Pooling and Dropout layers and artificially enlarging the training dataset through various image transformations.

In 2013, ZFNet[8] got the first place of ILSVRC with a top-5 error rate of 14.8%. ZFNet[8] was based on AlexNet[7] and it managed to improve the results by tweaking and optimizing the hyper parameters of AlexNet[7].

In 2014, GoogleNet[9] won ILSVRC[3] with a top-5 error rate of 6.67%. It consists of 22 layers, but due to the fact that it relies on very small convolutions, it has a significantly lower number of parameters compared to other networks. As an example, AlexNet[7] has 60 million parameters, while GoogleNet[9] has only 4 million parameters.

In 2015, ResNet[10] won ILSVRC[3] with a top-5 error rate of just 3.57% which can even beat human level performance on the ImageNet[2] dataset. This was possible by introducing the concept of skip connection which allowed to design much deeper models without introducing issues in training such as the vanishing gradient problem. In fact, one version of ResNet[10] consists of a total of 152 layers, which is several times larger compared to models that were introduced in the previous years.

Although, more models have been introduced in more recent years, both for image classification and for object detection, such as YoloNet[11], we won't be focusing on them in this thesis. Instead, the main focus will be on VGGNet[1], another deep Convolutional Neural Network that was introduced in ILSVRC 2014[3] and managed to get second place on the classification task with a top-5 error rate of 7.3%.

The idea behind VGGNet[1] was to improve the architecture of previous networks such as AlexNet[7] by stacking more convolutional layers. This was made possible by using smaller convolutional filters compared to the filter size that was used on previous models. For that reason, VGGNet[1] uses almost entirely convolutional layers with a filter size of 3×3, which is much smaller compared to the filter sizes of 7×7 or even 11×11 that were used in previous architectures such as AlexNet[7]. In fact, as explained on the VGGNet paper[1] as well, stacking three convolutional filters of size 3×3 with a stride step set to 1 can provide the same receptive field compared to a single 7×7 filter. The main benefit of using three layers instead of one is that the total number of parameters is $3x(3^2C^2) = 27C^2$ compared to $7^2C^2 = 49C^2$, where C is the number of channels found on the convolutional layers. This helped create and test much deeper architectures compared to previous ones, which performed significantly better on ILSVRC[3].

In total, six different configurations of VGGNet[1] were created, A to E, along with A-LRN which is the same as A but with some additional normalization layers. The layer information of all configurations can be seen on Table 1.

Table 1: VGGNet Configurations

A	**A-LRN**	**B**	**C**	**D**	**E**
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
Input (224×224 BGR image)					
Conv3-64	Conv3-64 **LRN**	Conv3-64 **Conv3-64**	Conv3-64 Conv3-64	Conv3-64 Conv3-64	Conv3-64 Conv3-64
Maxpool					
Conv3-128	Conv3-128	Conv3-128 **Conv3-128**	Conv3-128 Conv3-128	Conv3-128 Conv3-128	Conv3-128 Conv3-128
Maxpool					
Conv3-256 Conv3-256	Conv3-256 Conv3-256	Conv3-256 Conv3-256	Conv3-256 Conv3-256 **Conv1-256**	Conv3-256 Conv3-256 **Conv3-256**	Conv3-256 Conv3-256 Conv3-256

					Conv3-256
Maxpool					
Conv3-512 Conv3-512	Conv3-512 Conv3-512	Conv3-512 Conv3-512	Conv3-512 Conv3-512 **Conv1-512**	Conv3-512 Conv3-512 **Conv3-512**	Conv3-512 Conv3-512 Conv3-512 **Conv3-512**
Maxpool					
Conv3-512 Conv3-512	Conv3-512 Conv3-512	Conv3-512 Conv3-512	Conv3-512 Conv3-512 **Conv1-512**	Conv3-512 Conv3-512 **Conv3-512**	Conv3-512 Conv3-512 Conv3-512 **Conv3-512**
Maxpool					
FC-4096					
FC-4096					
FC-1000					
Softmax					

As it can been from Table 1, all configurations follow a similar format since each configuration was based on the previous one, but with more convolutional layers stacked. This resulted in architectures ranging from 11 layers to 19 and from 133 million parameters to 144 as it can be seen on Table 2.

Table 2: VGGNet Number of Parameters

A	A-LRN	B	C	D	E
133 million	133 million	133 million	134 million	138 million	144 million

In more detail about the networks themselves, we can observe the following. They all receive a 224×224 image as input, that gets processed by a stack of 3×3 convolutional layers which are separated by a total of 5 softmax layers. Afterwards, two fully connected layers with 4096 channels each can be found, followed by another fully connected layer with 1000 channels, since the models were trained for a total of 1000 classes. The final layer in all configurations is a softmax layer.

In all configurations, the filter size of the convolutional layers is 3×3, with the exception of configuration C where a filter size of 1×1 was also tested. The convolution stride is set to 1 pixel. The pixel window for the maxpooling layers is 2×2 and the stride is also set to 2. All hidden layers in the network are followed by ReLU (Rectified Linear Unit) functions, which can help with the vanishing gradient problem and improve performance.

We won't be focusing on the training and evaluation procedure of the above networks, since we'll describe how those steps were achieved in our own pipeline for the purposes of the this Thesis in Part 2. It should be noted, however, that the two best performing configurations in ILSVRC[3] were D and E which we will be referring to as VGG16 and VGG19 in the next parts. Specifically, VGG16 achieved a top-5 error rate of 7.2% and VGG19 7.1%. The combined output of these models was also tested by averaging the output of the softmax layer of the two networks, which achieved a top-5 error rate of 7.0%. Finally, a top-5 error rate of 6.8% was achieved, by combining again those two configurations, but through a different training process.

1.3. Our Objective

The aim of this Thesis is to create a pipeline that utilizes VGGNet[1] in order to detect vehicles found in CCTV footage, which is an object detection task since we need to extract multiple bounding boxes and their labels, per image.

For the purposes of this Thesis, we will use the two best performing configurations of VGGNet[1], VGG16 and VGG19. We will go through the data preparation, the training process of those two networks, see how we can use a network trained for classification for an object detection task through the extraction of region proposals with two different methods, Sliding Windows and Selective Search[12], look into the post-processing steps required and, finally, evaluate our pipeline with two different metrics, accuracy and mean Average Precision (mAP)[13].

2. SYSTEM OVERVIEW

In this section, we will see in greater detail how the vehicle detection system works, although most of the steps described below could be applied on different datasets and object detection problems as well. Our pipeline was created in Python3 with the help of Keras[14], a popular Deep Learning framework. All training and testing was done on an Nvidia Tesla T4 GPU in order to speed up the calculations.

Dataset

2.1.1 Overview

As mentioned already, the dataset that we will be using is CCTV footage with the goal of detecting all vehicles. More specifically, the dataset consists of 100.000 RGB images, sized 704×576, meaning that although we're not working with actual video footage, we are using the video frames themselves.

Those images are taken from multiple different cameras and in various times throughout multiple days. This results in a wide variety of different images, taken at different times of the day, so our system should be able to detect vehicles in both daylight and in the night. Additionally, the number of vehicles in the images can range from no vehicles to tens of them included in a single image.

Due to the variety in camera positioning and the high speed of vehicles that were moving while some of the pictures were taken, the images contain vehicles from various distances, meaning that some vehicle have lots of details present, while others cover only a very small portion of the total image and can even be shaky due to their movement.

Another important thing to point out about the images, is that the different weather conditions can result in image distortions. For example, rain drops on the camera or the sunlight hitting directly the camera can distort the image quality and make it harder to distinguish the vehicles in it.

Finally, most of the images contain some labels on the four corners, which contain the date, time and day the image was taken along with the number of the specific camera that used to take the image. This can also result in parts of vehicles being covered by the label text.

From the above, we can see that there is high variety in the images of our dataset. A few example images of the dataset can be seen on Image 1.

Image 1: Sample Images from Dataset

2.1.2 Data Preparation

It is also important to explain how we will use the aforementioned images in order to train our two Neural Networks. For our specific problem, which is detecting vehicles from provided footage, it should be pointed out that specifying the exact category of the vehicles is not important. Moreover, we do not aim to detect or provide bounding boxes for any non vehicle objects detected in the images. For that reason, we'll prepare a training dataset suitable for binary classification.

What this means is that we will train our two Neural Networks for just two classes. The first one will represent all vehicles, and it is the positive class since it represents the objects we want to detect and the second class will represent all non vehicles and will be the negative class, since it contains objects that we do not care about detecting.

For the creation of the training dataset, two different sources were used. The first one is a subset of the image dataset that we're working with since we want our network to be able to detect cars in situations like the ones described above. However, this dataset is not labeled, meaning that a lot of manual work was required in order to collect a high enough number of images.

The vehicle images were collected by taking a subset of the dataset, manually adding bounding boxes and then extracting those images. Considering the wide variety of the dataset, it is important to have a high number of training images that can represent a high number of real world scenarios. For that reason, a total of 5000 images of vehicles were extracted. Of course, it should be pointed out that an even higher number of training images could have been extracted, but due to the amount of time required to manually extract those images, we decided not to continue expanding the training dataset this way.

Instead, in order to further increase the size and variety of the vehicles in the training dataset, we used the Stanford Cars Dataset[15]. This dataset contains a total of 16.815 images split into 196 classes. Since we do not care about the classes defined in the Stanford Cars Dataset[15], we simply added a subset of this dataset into our own. More specifically, we added an additional 2000 images to our own training dataset. It should also be mentioned that the Stanford Cars Dataset contains images that are usually more high resolution compared to the ones that were manually extracted. Although all images will be resized to 224×224, a step that is described later, having images that are more clear can help with the training process and improve the feature extraction for the vehicle class.

Looking at Image 2 and Image 3, the difference between the vehicles found in our own dataset and in the Stanford Card Dataset[15] becomes immediately apparent. Even so, the addition of those extra images can help the training process and reduce overfitting.

Image 2: Sample Images of manually extracted Vehicles

Image 3: Sample Images from Stanford Cars Dataset

The second part of the training dataset is the non vehicles. Due to the vagueness of this category, many ways of creating such training dataset could be used. What we did is to collect a set of images similar to the ones that we might find in a testing dataset or in a real world scenario, excluding though all vehicles. Since we already added labels to a significant subset of the original dataset with the images from the CCTV footage, we can now use an algorithm called Selective Search[12] on those images in order to extract certain regions out of it. Selective Search[12] is described in a next section, but for now it should be mentioned that it is one of the region proposal methods used during testing, so using the same method in order to construct the training dataset for non vehicles can make a good strategy. After we receive the region proposals from Selective Search[12], we can then use the labels that were manually added in order to get rid of any images containing vehicles or even parts of vehicles. The result is the second part of our training dataset containing all the non vehicles.

Although it is not apparent from the above description, due to the high number of regions received from Selective Search[12], and due to the fact that the non vehicle objects we can run into during testing can have a very high variance, the non vehicles class of our training dataset has a much higher number of images compared to the vehicles class. Specifically, our training dataset contains 40000 images of non vehicles, which is almost 7 times as many as the vehicle images. This means that our training dataset is imbalanced and that could result in problems during the training of the neural networks. This, however, can easily be tackled by setting weights to the two classes in order to counter balance the different in number of images, while also keeping the benefits of having a high variety of non vehicle images.

We can also see from Image 4 some of the non vehicle images of the training dataset. Some of the non vehicle objects can be traffic signs, parts of the road, traffic lights and even the labels that are found on the corners of the images. We can expect to find similar images during testing as well.

Image 4: Sample Images of non Vehicles

Training

2.2.1 Finetuning VGGNet

As mentioned already, for this Thesis, we will be focusing on two of the six VGGNet[1] configurations, VGG16 and VGG19, since those two were the best performing ones in ILSVRC[3]. However, due to the high number of parameters that those two networks have, with VGG16 having 138 million parameters and VGG19 having 144 million parameters, and in combination with the relatively low number of images in our training dataset it is not feasible to train those Neural Networks entirely from scratch. Additionally, training Neural Networks with such a high number of parameters from scratch requires a high amount of epochs and thus becomes very time consuming. For those reasons, we looked instead into finetuning the two networks.

What this means is that we first load some other weight files into the layers of VGG16 and VGG19. Those weights were extracted by training the two networks with the ImageNet Dataset[2], which is significantly larger compared to our own training dataset and can provide a good starting point for finetuning the two networks for our own task.

Before, however, we can begin the finetuning process, a change is required on both networks. As described above, the last fully connected layer of all VGGNet[1] configurations contains 1000 channels. That is because the training dataset that was used to train those networks contains a total of 1000 classes. Since our own training dataset contains only 2 classes, we remove that layer and instead replace it one fully connected layer that only contains 2 channels. The rest of the layers in both networks, including the softmax layer that is right after our new fully connected layer, remain the same as before.

Moreover, for the finetuning process, one choice needs to be made. That is, the amount of layers to be finetuned, meaning that we can pick which layers' weights will be updated during our own training process and which ones will retain their values from the weights that we load from the pre-trained models. Although it is fairly common to update the weights of only some of the last layers of a Neural Network, we instead opted to finetune all layers, both convolutional and fully connected. This is because, while we were evaluating the training process, we concluded that finetuning all layers provides better results compared to finetuning only some of them. This actually comes in agreement with the original VGGNet Paper[1], where the models that were trained with ImageNet[2] were finetuned with additional training datasets in order to evaluate the performance of those models for other tasks as well.

Regarding the training parameters that were used, they are mostly similar to the ones used to train the original VGGNet[1] on ImageNet[2]. The main differences is the initial learning rate that was set to 10^{-3} instead of 10^{-2} and the batch size that was set to 64 instead of 256. The first one was set to a lower value so that we can still retain the values of the weights that were loaded during the finetuning process, since otherwise a higher learning rate would result in completely overriding those values and resulting in our models being unable to benefit from using the weights from the pretrained models. As for the batch size, there is no actual reasoning behind this change other than memory limitations. Even so, the lower batch size was still high enough to be able to finetune the two models. As for the rest of the parameters, momentum was set to 0.9, the dropout ratio was set to 0.5 and the weight decay was set to 5×10^{-4}. We also set the class weight ratio to be 1/7 so that we counter the issue of having an unbalanced training dataset and successfully train the models for both classes without overfitting.

Although we used Stochastic Gradient Descent (SGD) as the optimizer for the training process, but we also tested the Adam optimizer[16]. We noticed, however, that our trained networks failed to generalize when trained with the Adam optimizer[16] and thus, we ended up training our final models with SGD and the parameters that were mentioned above. It seems, from related research, that adaptive methods cannot generalize well enough for somewhat simple problems, such as binary classification, which would explain why using SGD provided better results during training[17].

2.2.2 Preprocessing

The preprocessing of the images before we provide them to the Neural Networks, is the same as the one that was done on the original, pre-trained models. We first calculate the mean RGB value for the training dataset, by using all the pixels per each color channel and substract it from the image, so that it can have both positive and negative values. By centering the values of the images around zero, we improve and speedup the training process. We additionally divide all values in the images by 255 so that the values are now set between -1 and 1 as using floats instead of integers values can help improve the performance and accuracy of calculations during the training process of the two Neural Networks. We also convert the images from RGB to BGR since that was the format that was used for the initial training in the VGG Paper[1]. Finally, all images are converted to 224×224, since that was the same size that was used during the initial training. Additionally, before each epoch, we apply some data augmentation techniques on the entire training dataset through certain transformations such as horizontal flipping, small shifts in RGB values and brightness of the images and taking random crops from the images in order to increase the variety of the dataset and reduce overfitting.

For the training process, we split our dataset into parts, one that is for training, which consists of 80% of the dataset and one for validation which consists of the other 20%. The splitting was done in a way so that both the training and the validation datasets have the same class ratio as the original dataset. This means that the training dataset was made with 80% of the vehicles we had collected and 80% of the non vehicles. Similarly, the validation dataset was made with 20% of the vehicles we had collected and 20% of the non vehicles. This way, we can still use the class weight ratio that was mentioned above for the training process.

2.2.3 Training Metrics

During the training process, we calculated four metrics in order to evaluate the process, the accuracy and the categorical cross entropy loss for both the training and the validation dataset. These metrics are used in order to determine how well our models can classify the given datasets and to determine whether or not our model has avoided overfitting so that it can be used on unseen data.

Both models were trained for a total of 20 epochs in order to avoid overfitting. On Figure 1 and Figure 2, we can see the metrics from training VGG16 and VGG19 respectively.

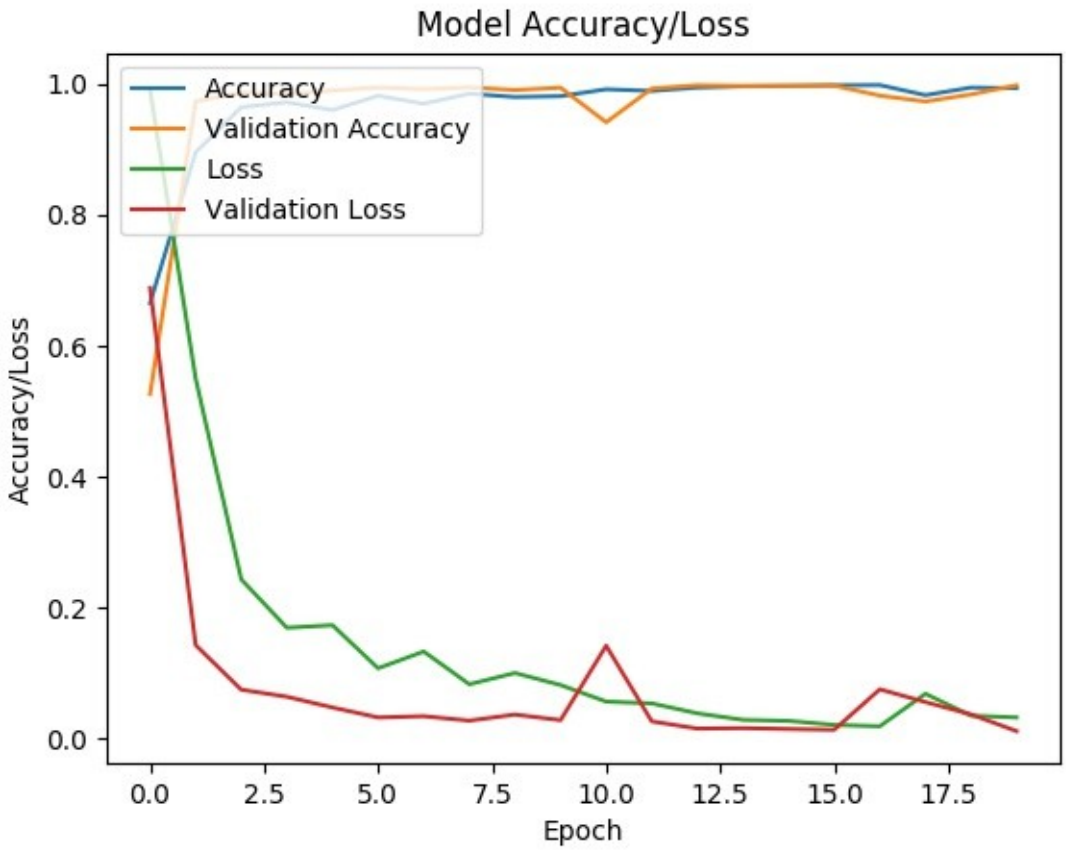

Figure 1: VGG16 Training Metrics

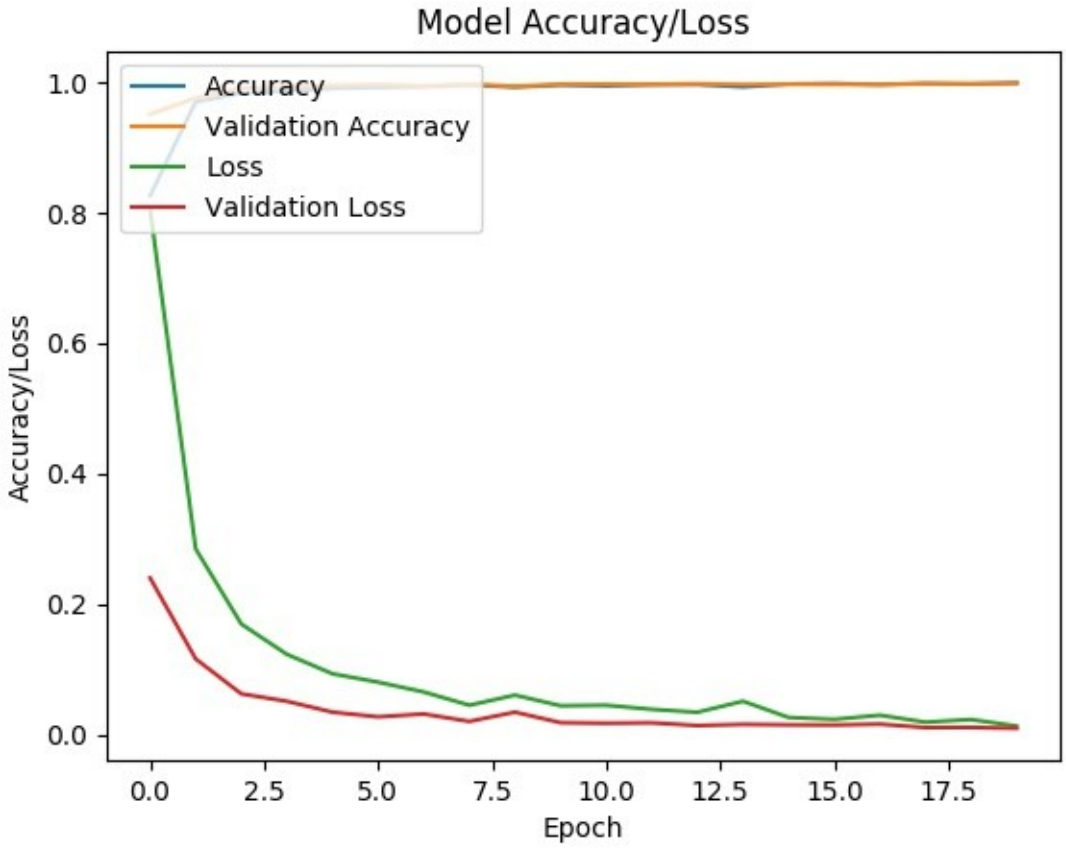

Figure 2: VGG19 Training Metrics

From the two figures, we notice that all metrics stop improving at about 20 epochs for both models, which is why we stop the training process there. Since our validation accuracy and validation loss metrics have both improved during the training process, we can assume that our networks have managed to learn important features from the training dataset and have avoided overfitting so that we can use them to extract labels from unseen data.

Region proposals

2.3.1 Need for Region Proposals

Now that the two Neural Networks have been trained, we need a way to use them in order to extract bounding boxes instead of just labels. Since we want to detect multiple vehicles from a single image and the networks that we trained are suited for classification, we cannot just provide the testing images as they are. What we can do instead is to provide region proposals.

What this means is that we can extract parts of the testing images and provide them to our trained networks in order to get a label for them. Since our networks are trained so that they can identify vehicles, this means that if the extracted image is labeled as a vehicle, we can use that information, along with its location of the region proposal on the original testing image in order to draw the bounding box. This way, it becomes possible to use a Neural Network trained for classification for object detection tasks.

2.3.2 Sliding Windows

However, what becomes an immediate issue is how we can actually extract those region proposals. One simple way of doing so is through Sliding Windows. The way it works is that we have a window, which is a subset of the whole image and a step size. We move the window according to the step size vertically until we go from one side of the image to the other. We then place the window back to the original horizontal position, but also move it vertically according to the step size and then start moving it horizontally again. We continue this process until the whole image has been covered. While the window is "sliding" on the image, we extract that part of the image and provide it to the Neural Networks for classification.

Although what was described above is the typical way of using Sliding Windows, we instead opted for a somewhat different method. The reasoning behind that is that we wanted a better control of the size, aspect ratio and number of region proposals to be extracted, while the step size itself doesn't not affect the results, since it's merely a parameter to be used for the extraction of the windows. The size and aspect ratio are important since from the training dataset we can see that extracting windows of a specific size and aspect ratio could help us extract better regions that have a higher possibility of including vehicles. This way, we can reduce the number of region proposals to classify, improve performance and reduce the potential false positives during the classification of those regions.

Our modified version of Sliding Windows receives a list of numbers, which define one side of the window, a list of aspect ratios and a number that defines how many region proposals we would want to generate. We use those parameters to determine the step size that would be needed in order to approximate the desired number of region proposals and then we generate windows from all combinations of sizes and aspect ratios that we provided.

In order to calculate the step size, we will need certain parameters: the *number of windows* that we want to generate, a list of *sizes* which contains the size of one side of the region proposals that we want to extract and a list of *ratios*, which defines what aspect ratios we want our regions proposals to have. For example, for *sizes=[10]* and *ratios=[[1,1],[1,2]]* we will end up with a number of region proposals of sizes *[10, 10]* and *[10, 20]*.

The calculation of the step size per each window size that was provided, is the following:

$$steps\ \ per\ \ window = \left\lfloor \frac{number\ \ of\ \ windows}{number\ \ of\ \ sizes\ \times\ number\ \ of\ \ ratios} \right\rfloor$$

$$step\ \ sizes = [\,]$$

$$for\ \ size\ \ in\ \ sizes:$$

$$\text{step sizes} \Leftarrow \left\lfloor \frac{\sqrt{image\ \ height\ \times\ image\ \ width}}{\text{size}\ \times\ \text{steps per window}} \right\rfloor$$

2.3.3 Selective Search

Although this modified method of Sliding Windows allows us to extract region proposals in a more smart way, since we're adjusting some of the parameters to our specific problem we're trying to solve, it's still a very simplistic way of getting those regions, since it does not take under consideration any of the information in the images themselves. Additionally, Sliding Windows is an exhaustive search for regions and generating proposals at various sizes and aspect ratios can quickly result to a very high number of images to classify.

What we would prefer instead is a method of generating a small number of windows that can successfully find the objects of interest within our images, which in our case is vehicles. This method should also be able to generate region proposals at various sizes and shapes without actually having to specify these parameters. It is important for such method to have a high recall value, so that our Neural Networks can successfully classify the vehicles. A lower recall value could help reduce the number of false positives, but it would also impact negatively the accuracy of our system, so we would prefer to avoid that. There are several such algorithms that can do what was described, but we will be focusing on Selective Search[12].

Selective Search[12] is a region proposal algorithm designed to be fast and provide a high recall value. It works by segmenting the provided image into very small regions and then using certain criteria in order to combine those regions into larger ones, which can then be used to retrieve bounding boxes. Those boxes are the region proposals that we use.

In greater detail, Selective Search[12] first over-segments the image by using the graph-based segmentation algorithm by Felzenzwalb and Huttenlocher[18]. Providing a great segmentation result is not the goal of this method, since at this stage our objects of interest, which are vehicles, are split into many small segments. However, Selective Search[12] uses those generated segments as a starting point and through multiple iterations, it merges adjacent segments based on certain similarity measures. Eventually, larger segments which are actually objects found in the original image are created. We can then convert those segments to bounding boxes, extract them and use the trained Neural Networks to classify them.

The similarity measures that Selective Search[12] uses in order to combine adjacent segments are the following four:

1. Color similarity: A color histogram of 25 values is calculated for each color channel of the image. The histograms for all channels are then concatenated in order to obtain a color descriptor with a total of 75 dimensions.
2. Texture similarity: Texture features are calculated by extracting Gaussian derivatives at 8 orientations for each color channel. For each orientation and for each color channel, a histogram of 10 values is computed resulting into a texture descriptor with a total of a 240 dimensions.
3. Shape compatibility: Measures how well two regions fit into each other. If one region fits into another region, we would want to merge them in order to fill any gaps. If the regions are not adjacent, we do not want to merge them at all.
4. Size similarity: Helps ensure that region proposals at all scales are formed at all parts of the image. Without this measure, a single region would be combined with all the smaller adjacent regions one by one and hence region proposals at multiple scales would only be generated at this location.

The final similarity measure is a linear combination of the above four metrics. Selective Search[12] can also be further customized through three parameters:

1. Sigma: Affects the results of the segmentation step.
2. Base K: Affects the size of the first segments.
3. K increment: Affects how segments of larger sizes are created.

Since the above three parameters can greatly affect how Selective Search[12] generates region proposals, we needed to set values that would provide a high recall value for our specific dataset. For that purpose, we used Selective Search[12] on the entire training dataset in order to extract region proposals. We then used Intersection over Union (IoU), a metric that is described later, which can help us compare the region proposals with the vehicles that were manually labeled and see if there is an overlap between those regions.

We eventually set sigma to 0.9, base K to 50 and the K increment to 250 since Selective Search[12] with those parameters managed to detect about 99% of all vehicles in our training dataset. Since the testing dataset is derived from the same camera footage, we can expect Selective Search provide similar accuracy on that as well.

2.4 Non Maximum Suppression

By using Selective Search[12], we now have an efficient way of providing region proposals that can include most of the vehicles, so our trained models can now classify them. After we receive the results from the networks, we can use the results of the softmax layer in order to add a label to each region proposal by picking the class with the highest confidence value. However, at this stage, another issue becomes apparent, both with Selective Search[12] and even with Sliding Windows, depending on the step size. The issue is that our region proposal methods can provide a lot of overlapping windows, meaning that our networks could classify the same vehicle multiple times and thus lead to a high number of false positives. This means that in order to solve this issue, we need a way to remove overlapping images.

Non Maximum Suppresion (NMS) is a popular way of doing so. What we do is to provide all regions of the vehicle class to it, along with the position of those regions in the original image and the confidence metric that we received from the softmax layer of our networks. We then start iterating over all the regions and we use Intersection over Union (IoU) in order to find which of those regions are overlapping.

IoU is simply a ratio where the numerator is the area of overlap between two regions and the denominator is the area of union between the same two regions. A value of 0 means that the two regions are not overlapping at all, while a value of 1 would obviously mean that the two regions are identical. Any values in-between can show us how much overlap there is between the two regions.

For NMS, we will consider that two regions have a high IoU value when it's 0.5 or above. When that happens, we compare the confidence between the two overlapping regions and remove the one with the lower value. This process continues until there are no more regions with an IoU equal or higher than 0.5. After NMS is completed, we now have our predictions for the testing dataset, but without any overlaps.

Evaluation

2.5.1 Testing Dataset

Up until this point, we only described how our object detection system works, the challenges that we faced and how we tackled them. We need, however, a testing dataset and certain metrics in order to be able to actually evaluate how well the described system works.

In order to create the testing dataset, we took 250 images from the original dataset, and manually labeled all the vehicles found on them in order to be able to evaluate the accuracy of our object detection system.

For the creation of the testing dataset, we tried picking images that could be able to represent as much of the whole dataset as possible, meaning that the final testing dataset contains images from many different times of the day, at different weather conditions and the number of vehicles ranging from none to over a dozen.

This way, we can certain that if our object detection system works well enough on the testing dataset, it would also be able to generalize enough so that it could be used for real, unlabeled data.

2.5.2 Accuracy

Now that we have described the testing dataset, we also need some metrics to be used along with it. The first metric to use is the Accuracy for the vehicle detection. Since there is a limited number of vehicles to detect, we want to calculate a percentage that shows us how many of those vehicles our system managed to detect. A low recall measure in our region proposal algorithms or a high number of false negatives in our networks could negatively affect accuracy.

In order to calculate the accuracy, we use the list of regions that were classified from the neural networks afters it's been filtered by NMS. We then we go through each of the vehicle predictions and the ground truth labels and calculate the IoU. After the calculation of the IoU, we need to set a threshold and compare the calculated value with that in order to decide whether the prediction can be considered as a correct prediction. A low threshold can result in our predictions having a relatively low overlap with the ground truth regions, but a very high threshold could significantly reduce the calculated by considering as correct predictions only those that have a very high overlap with the equivalent ground truth regions. For our testing, we considered 0.5 a balanced value for the threshold. An IoU of 0.5 means that the prediction covers 50% of the surface of the ground truth region while the ground truth region also covers 50% of the surface of the prediction.

Going back to the calculation of the Accuracy, if a prediction has an IoU higher than 0.5 we consider it correct, but we do not use the same prediction for other vehicles, meaning that each prediction can only be matched with one ground truth region. This makes sense, since we want each prediction to include exactly one vehicle.

2.5.3 Mean Average Precision

Although the Accuracy can help us see how many vehicles our system can detect, it does not give us any indication of how many false positives were done, or how our system handles non vehicle region proposals. For that purpose, we need another metric that can provide us with additional information. This can be achieved with mean Average Precision (mAP)[13], a common metric for evaluating object detection models that has also been used in the PASCAL VOC Challenge[13], a known challenge for object detection.

In order to calculate the mean Average Precision[13], the following steps are applied:

1. We separate the vehicle and non vehicle predictions that we receive from the two Neural Networks and apply NMS on each of them to remove the overlapping predictions.

2. We now need to calculate the true positives and false positives for the vehicles. In order to do that, we first sort the vehicle predictions based on the confidence value that we received from the softmax layer of the Neural Networks. This way, when we match a prediction with a ground truth label, we give higher priority to the ones with a higher confidence. We then go through each vehicle prediction and we calculate the IoU between the prediction and each ground truth region, while storing in a list all the predictions that the IoU has a value that is higher than our threshold. For mAP[13], the threshold is the same as the one we used for accuracy, so it's set to 0.5. When we find a vehicle prediction that has an IoU over 0.5, but that ground truth region has been matched already to a prediction with a higher confidence, we instead count that prediction as a false positive. Additionally, any predictions that did not match any of the ground truth labels are also counted as false positives. At the end, the matched vehicle predictions are the true positives, while the rest are the false positives. It should be noted, that during the NMS step, we calculate IoU between predictions in order to avoid overlaps. However, during this step, we calculate the IoU between a prediction and the ground truth labels. It is possible that two predictions have a low IoU value between them, but both of them have an IoU that is higher than our threshold, meaning that even though we removed many overlapping regions during the NMS step, we could still end up with multiple regions overlapping the same ground truth label.
3. We also need to calculate the true positives and false positives for the non vehicles. The process for doing that is similar to the one described for the vehicles, but with a few subtle, but important, changes. We first sort the non vehicle predictions based on the confidence value that we received from the softmax layer of the Neural Networks. We then go through each non vehicle prediction and we calculate the IoU between the prediction and each ground truth region, while storing in a list all the predictions that the IoU has a value that is higher than 0.5, which is our set threshold. This time, when we find a non vehicle prediction that has an IoU over 0.5, and that ground truth region has been matched already, we store it on the same list, unlike before with the vehicle predictions. At the end, the matched non vehicle predictions, are all false positives, since our ground truth labels contains only vehicles, while the predictions that were not matched with any of the ground truth regions are the true positives.
4. We then calculate the Average Precision (AP) for the vehicles and non vehicles the following way, where *TP* are the true positives for either the vehicle class or the non vehicle class and *FP* is the false positives:

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

5. Finally, mAP[13] can be calculated by simply averaging the *vehicle AP* and n*on vehicle AP*, meaning that in our case, it simply is:

$$mAP = \frac{Vehicle \quad AP \quad + \quad Non \quad Vehicle \quad AP}{2}$$

It becomes obvious from the above, that mAP[13] can help us evaluate how well our system works by giving us an overview of both classes. Since it takes under consideration both true positives and false positives, a high Accuracy, but low mAP[13] would indicate that our system results in too many false positives for the vehicle class. By combining the two metrics, we can have a better understanding of the weakness of our system, which is important for improving it.

Finally, another metric of interest is how long our system takes to detect vehicles from a single image. Although this does not help us evaluate the performance of the system, it can help us compare the various benchmarks that we did on both performance and time efficiency. It becomes especially useful when comparing the two different region proposal methods, Sliding Windows and Selective Search[12].

3. BENCHMARKS

In this section, we're going to describe various benchmarks, present the results, explain them and offer solutions and suggestions. For the various benchmarks, we will use the finetuned VGG16 and VGG19 networks with both Sliding Windows and Selective Search[12] for region proposals. We will also look into combining the output of VGG16 and VGG19, in a setup similar to the found in the original VGG Paper[1], which we will be calling FusionNet.

3.1 Sliding Windows and VGGNet

We will start by using our version of Sliding Windows as the region proposal algorithm, along with VGG16 and then with VGG19 as the classifiers of our system.

For Sliding Windows, we will first try using with *70×70* windows and about 3000 regions proposals, since we saw during initial testing that most vehicles in the training dataset were close to that size and with an aspect ratio that is close to *1:1*. We will also try using *42×42* and *96×96* along with *70×70* as those two extra sizes were also commonly found in the training dataset. The aspect ratio will remain the same for the second configuration of Sliding Windows, but the number of generated region proposals will get increased to 9000 in order to keep the step size the same and generate the same number of windows per size as with the first configuration. Finally, we will test Sliding Windows with the same set of windows, but with *1:1*, *1:2* ans *2:1* aspect ratios, which, based again on the training dataset, should help us detect more vehicles during testing. In order to keep the same number of region proposals per size, we also increase the number of regions to generate to 27000.

On Table 3, we can see the Accuracy, mAP[13] and required time to classify all region proposals per image for the aforementioned configurations with VGG16.

Table 3: Sliding Windows and VGG16

Regions	Accuracy	mAP@0.5	Time
~3000	39.78	66.13	14s
~9000	59.67	58.77	40s
~27000	81.57	57.79	122s

On Table 4, we can see the Accuracy, mAP[13] and required time to classify all region proposals per image for the aforementioned configurations with VGG19.

Table 4: Sliding Windows and VGG19

Regions	Accuracy	mAP@0.5	Time
~3000	37.09	67.58	17s
~9000	59.13	60.2	48s
~27000	80.67	58.95	143s

From our metrics with Sliding Windows, we can observe a few things. Firstly, we see that the first two configurations, with 3000 and 9000 regions provided a very low Accuracy value, with the first one failing to find even half of the vehicles from our test dataset. This can be attributed to the low recall value of Sliding Windows, since even when we use specific window sizes by examining our training dataset, we still couldn't find many of the vehicles with our region proposal algorithm. On the other hand, our third configuration with 27000 region proposals managed to achieve a fairly high accuracy, since both VGG16 and VGG19 scored above 80%, thanks to the high amount of region proposals with different sizes and ratios.

Regarding the performance of the models themselves, we see that the mean Average Precision[13] is relatively low for all configurations and for both models. This is probably due to the fact that our Sliding Windows algorithm has a low recall value, meaning that many of the region proposals are of non vehicle objects and thus the true positives are low, while the false positives are potentially increased. This can be confirmed by looking at how mAP[13] gets even lower when we provide a higher number of region proposals. Specifically, considering the fact that the vehicles that can be found in a single image are at most a couple of dozens, generating a total of 27000 region proposals is a very high number, meaning that many of those regions are either non vehicles or overlapping vehicles. This can very easily lead to a very high number of false positives, which is why although we see our accuracy increase, the mean Average Precision is in fact dropping.

Additionally, by looking at our two different models, we notice that there are some small differences. VGG16 consistently provides a higher accuracy, while VGG19 provides a higher mAP[13]. Although, the models were trained on the same dataset, the difference in their architecture and the slight differences that could result from the image transformations that we applied on every training epoch, mean that we can't expect the result of these two models to be identical. From the metrics we have so far, we notice that VGG16 can detect more vehicles, at the cost of more false positives, while VGG19 is less likely to predict that a region is indeed a vehicle.

Finally, from our time metric, we can see that the time required to detect vehicles from just one image is rapidly increasing when moving from one configuration, while VGG19, as expected due to its higher number of layers and complexity, is consistently slower than VGG16. This makes it obvious that using the best configuration of Sliding Windows is any real world scenario is not feasible, since a waiting time of about 2 minutes for a single frame would make the system impractical to use on a video source, even with more powerful hardware than what was used for testing.

3.2 Selective Search and VGGNet

As we saw above, Sliding Windows failed to deliver good results on all three metrics that we used. Using a small number of region proposals might be somewhat fast, but it comes at the cost of having a low Accuracy. Increasing the number of regions can provide a good accuracy, but it reduces the mean Average Precision[13] and significantly increases the time required to retrieve the results.

We will now look into using Selective Search[12] as the region proposal algorithm in order to improve both Accuracy and mAP[13] while also reducing the amount of regions to classify, meaning that we expect our system to be much faster compared to using Sliding Windows.

We will use both the Fast and the Quality versions of Selective Search[12]. The first one, as the name implies, offers better performance, but has a lower recall value and thus provides a lower number of region proposals. For both versions, we will be testing with 1000, 2000 and 3000 regions proposals.

Regarding those numbers, there are two things to point out. First, we didn't use the Fast version of Selective Search[12] with more than 3000 regions since during our testing we noticed that Selective Search[12] did not generate as many region proposals for any of the testing images. As for the Quality version of Selective Search[12], although for some images it generated as many as 10000 region proposals, using more than 3000 regions did not improve the accuracy in our initial testing, while it lead to an increase in false positives due to the increased number of overlapping regions and non vehicle regions.

On Table 5, we can see the Accuracy, mAP[13] and required time to classify all region proposals per image for the aforementioned configurations with VGG16.

Table 5: Selective Search (Fast) and VGG16

Regions	Accuracy	mAP@0.5	Time
1000	56.45	80.37	6s + 4s
2000	75.8	74.86	11s + 4s
3000	80.1	72.24	15s + 4s

On Table 6, we can see the Accuracy, mAP[13] and required time to classify all region proposals per image for the aforementioned configurations with VGG19.

Table 6: Selective Search (Fast) and VGG19

Regions	Accuracy	mAP@0.5	Time
1000	59.67	82.12	7s + 4s
2000	72.35	75.16	13s + 4s
3000	77.9	73.3	19s + 4s

On Table 7, we can see the Accuracy, mAP[13] and required time to classify all region proposals per image for the aforementioned configurations with VGG16.

Table 7: Selective Search (Quality) and VGG16

Regions	Accuracy	mAP@0.5	Time
1000	63.58	81.9	6s + 9s
2000	78.94	76.27	11s + 9s

3000	82.01	73.8	15s + 9s

On Table 8, we can see the Accuracy, mAP[13] and required time to classify all region proposals per image for the aforementioned configurations with VGG19.

Table 8: Selective Search (Quality) and VGG19

Regions	Accuracy	mAP@0.5	Time
1000	65.74	82.3	7s + 9s
2000	77.94	77.62	13s + 9s
3000	83.55	74.29	19s + 9s

From our metrics with Selective Search[12] we can observe a few things. Firstly, we see that the first configuration, with just 1000 regions provided a relatively low accuracy value, but with the highest mAP[13] we've seen so far. This can be attributed to the high recall value of Selective Search[12], since even with a much smaller number of region proposals compared to Sliding Windows, we can achieve much better results. In fact, our system with Selective Search[13] and just 1000 region proposals managed to achieve a similar or higher Accuracy compared to the second configuration of Sliding Windows, which used a total of 9000 region proposals.

On the other hand, our second and third configurations with 2000 and 3000 region proposals respectively, managed to achieve a high accuracy, with the later one coming close or even surpassing the accuracy of the best configurations that were tested with Sliding Windows. This shows us, that by using a region proposal algorithm that can take advantage of the actual content of the image in order to generate regions, can provide similar accuracy compared to exhaustive methods such as Sliding Windows, even when the amount of generated regions is several times smaller.

The above, can also help improve the mean Average Precision[13] of our system. More specifically, we can see that mAP[13] has a fairly high value for all configurations and for both models. This can once again be attributed to the high recall of Selective Search[12], since by generating a much smaller number of region proposals which contain a high number of vehicles, we can increase the accuracy of our system while also significantly decrease the potential number of false positives.

However, similarly to the Sliding Windows results, we notice again that although the accuracy is increasing when we increase the number of generated region proposals, mAP[13] is actually getting lower. The reasoning behind this is the same, as before. By classifying more images, where only a subset of them contains vehicles, we increase the number of false positives and thus reduce the value of mAP[13]. The reason why Selective Search[12] can give us a much higher mAP[13] is because it can generate a much smaller number of region proposals and thus we end up with less overlapping vehicles and non vehicles.

Additionally, by looking at our two different models, we notice that there are some small differences, which are similar to the ones we found when we tested our system with Sliding Windows. VGG16 consistently provides a higher accuracy, while VGG19 provides a higher mAP[13]. Once again, this difference can be attributed to the different architecture of the two Neural Networks and the small difference during training due to the data augmentation techniques that we used.

We also notice that the Quality variant of Selective Search[12], although much slower, provides a somewhat higher Accuracy and mAP[13]. This is expected as the Quality version of Selective Search[12] is supposed to provide a higher recall value, meaning that our regions proposals, although being the same in terms of numbers compared to the Fast version, contain a higher number of vehicles, thus allowing our model to detect more vehicles and provide less false positives. Despite its increased overhead, it becomes clear that in terms of Accuracy and mean Average Precision[13], the Quality version of Selection Search[12] provides the best results of all region proposals algorithms and configurations that we've tested so far.

Finally, from our time metric, we can see that the time required to detect vehicles from just one image is significantly increasing when moving from one configuration, while VGG19, as expected due to its higher number of layers and complexity, is consistently slower than VGG16. We also notice that Selective Search[12], and especially the Quality variant, adds a noticeable overhead that wasn't present when we used Sliding Windows, which is a much simpler algorithm to execute. Despite that, however, we notice that although our system still cannot handle real time data, it is significantly faster than when we use Sliding Windows, with some configurations being nearly 5 times faster. This, while also taking under consideration the fact that with Selective Search[12] we managed to achieve a higher Accuracy and a much better mAP[13], makes Selective Search[12] a more suitable region proposal method for our object detection system.

3.3 Selective Search and FusionNet

Since Selective Search[12] provides better Accuracy and mean Average Precision[13] compared to Sliding Windows, while also improving the classification time by generating less region proposals, we will now look into further improving the results by combining the output of the two Neural Networks. This is done by averaging the output of the softmax layer of the two networks and using that as the final result. This process does not require retraining our two networks, so we can reuse the ones that were trained and tested already. The idea behind this is that by combining the output of the two networks, the accuracy increases while the number of false positives can also drop, due to the complementarity of the models. It is also common to combine the output of even more models, or train the same architecture on different subsets of a training dataset, but due to the time required to train and test multiple models, we did not test more complex combinations of our models.

For our testing with FusionNet, we chose not to use Sliding Windows, since it offers a much lower mAP[13] compared to Selective Search[12] while also requiring a high amount of time in order to be tested. Instead, we used Selective Search[12], both the Fast and the Quality variants.

On Table 9, we can see the accuracy, mAP[13] and required time to classify all region proposals per image for the aforementioned configurations with FusionNet.

Table 9: Selective Search (Fast) and FusionNet

Regions	Accuracy	mAP@0.5	Time
1000	61.79	83.31	13s + 4s
2000	77.62	76.91	24s + 4s
3000	82.16	74.49	33s + 4s

On Table 10, we can see the accuracy, mAP[13] and required time to classify all region proposals per image for the aforementioned configurations with FusionNet.

Table 10: Selective Search (Quality) and FusionNet

Regions	Accuracy	mAP@0.5	Time
1000	64.65	83.82	13s + 9s
2000	80.16	78.95	24s + 9s
3000	84.87	75.68	33s + 9s

From our metrics with FusionNet, we can, once again, observe a few things. Firstly, just like before, we notice that using a lower number of region proposals results in a lower accuracy, but a higher mAP.[13] We also notice that both Accuracy and mAP[13] are higher when we use the Quality version of Selective Search[12] instead of the Fast one. These observations are consistent with previous benchmarks, since we're using the same region proposal algorithm with the same parameters.

We also notice that, our FusionNet model can provide a slightly higher Accuracy and mAP[13] for all configurations of both versions of Selective Search[12]. This comes in agreement with the opinion shared above, that combining models with different architectures that were trained under somewhat different conditions can improve classification results, since the averaging of the confidence values of the two models can help eliminate false positives that one model may do that the other won't. The difference in the results of each model becomes apparent when we notice that VGG16 provides a higher Accuracy, while VGG19 provides a higher mAP[13], so the combination of the results of the two networks can result in an increase of the confidence value of correct vehicle detections and a decrease of the confidence value of incorrect vehicle detections, thus resulting in better metrics.

As expected, we notice the required time for the object detection is significantly increased due to the overhead of Selective Search[12], but also because we have to classify each batch of region proposals twice, once with VGG16 and once with VGG19. Despite that, we notice that even when we use FusionNet, the total required time is several times lower compared to the one from the Sliding Windows configurations and only a few seconds slower compared to using only one Neural Network.

Finally, from the metrics that we saw, we can conclude that the best object detection configuration is using FusionNet with the Quality version of Selective Search[12] at 3000 region proposals. It provides an Accuracy of 84.87% and a mAP[13] value of 75.68%. Although it is possible to get a better mAP value, it comes at the cost of greatly reducing the Accuracy, which is why we'll be using the aforementioned configuration for our next part.

3.4 Comparison with state-of-the-art: YoloNet3

As mentioned at the beginning, several Deep Learning models have been designed and published over the past years that are capable of either classifying images or detecting objects inside an image.

Just like our system, many of these models utilize region proposal methods in order to classify parts of the original image and provide labels and bounding boxes for those regions. One modern Deep Learning model that works differently is YoloNet[11] an object detection Neural Network that focuses on being fast by detecting objects from single image instead of region proposals.

YoloNet[11] trains on whole images for the purpose of detecting objects and thus has a better context during testing time regarding the appearance of objects within an image and their background. As such, it learns generalizable representations of objects and provides a lower number of false positives, while also being much faster compared to more traditional architectures.

Although we won't be focusing further more on the architecture of YoloNet[11], we will use its latest version, YoloNet3[2], and compare it with our best object detection system in order to have the opportunity of comparing our designed system with a state-of-the-art, pretrained model suited for object detection.

We won't look into training or finetuning YoloNet3[2], as this is out of the scope of Thesis. The pretrained model was trained using ImageNet for the purposes of object detection. Since ImageNet[2] contains a high number of several that belong to vehicles, such as "car" and "truck", we can expect the pretrained YoloNet3[19] to be able to detect vehicles from our testing dataset as well.

On Table 11, we can see the Accuracy, mAP[13] and required time for our best system and YoloNet3[19].

Table 11: Comparison with YoloNet3

Model	Accuracy	mAP@0.5	Time
Our Model	84.87	75.68	41s
YoloNet3	93.56	89.78	<1s

As we can see, the pretrained YoloNet3[19] outperforms our model, both in Accuracy and mean Average Precision[13], meaning that not it detects more vehicles, but it also manages to do less false positives. Additionally, thanks to its architecture that does not require generating region proposals, it is also much faster compared to our model since it requires less than a second in order to detect vehicles from a single image.

Although it becomes obvious that in a real world scenario, YoloNet3[19] would be more suitable to due the better metrics and faster performance, it was also noticed that in certain cases it made false positives or missed vehicles. As we mentioned earlier where we were describing the dataset that we worked with, the images are of low resolution and many vehicles appear shaky due to their movements. Additionally, many vehicles are partially hidden behind the labels that are found on the four corners of the images. We noticed while comparing YoloNet3[19] with our model that failed to detect several vehicles in those situations while it also made some false positives in case of low quality images.

This shows that even though its overall results are better than our model, there are ways of potentially improving it even further. It's apparent, that even though the original training dataset contains a wide range of vehicle images, they are not suitable to train a model to handle some difficult situations and edge cases found in our testing dataset. It is estimated that finetuning YoloNet3[19] with a training dataset similar to the one we created for this Thesis would help improve its results for such cases.

In any case though, this comparison between VGGNet[1] and YoloNet3[19] showed us how much Deep Learning models have improved over the last years, since YoloNet3[19] was published in 2018, just 4 years after VGGNet[1].

4. CONCLUSION

In this Thesis, we designed a system capable of detecting vehicles from images by using VGGNet[1]. We created datasets for training, validation and testing and looked into the proprocessing steps required for the images before they are provided to the Neural Networks. We finetuned two of the VGGNet[1] variants, VGG16 and VGG19, and optimized the training parameters in order to avoid overfitting. We also looked into two different ways of providing region proposals to our trained models, Sliding Windows and Selective Search[12]. We used NMS as a post processing step in order to reduce the amount of false positives and evaluated our system with Accuracy and mAP[13].

We described the differences in the results between VGG16 and VGG19 and noticed that Selective Search[12] provides better results both in Accuracy and mAP[13] with less region proposals compared to exhaustive search methods such as Sliding Windows. We also combined the output of the two training networks in order to further improve the results of our system. Finally, we compared our best system a modern, state-of-the-art Neural Network, suitable for object detection tasks, YoloNet3[19], which confirmed our initial statement on the vast improvement in computer vision tasks thanks to new and constantly better Deep Learning architectures being made and published.

Future directions include, but are not limited to, expanding our training dataset in order to improve the training process of our networks, training more variations of VGG16 and VGG19 by using subsets of our dataset and combining those models, thus creating a system that uses more than one or two classifiers and testing different region proposal algorithms that could provide a recall value similar to Selective Search[12], but with an even lower number of region proposals in order to improve the classification time and improve the mean Average Precision[13] of our system. Finally, some other steps could include using our system on different object detection tasks and comparing its performance on those with its performance on vehicle detection and comparing our system, with other modern Deep Learning models.

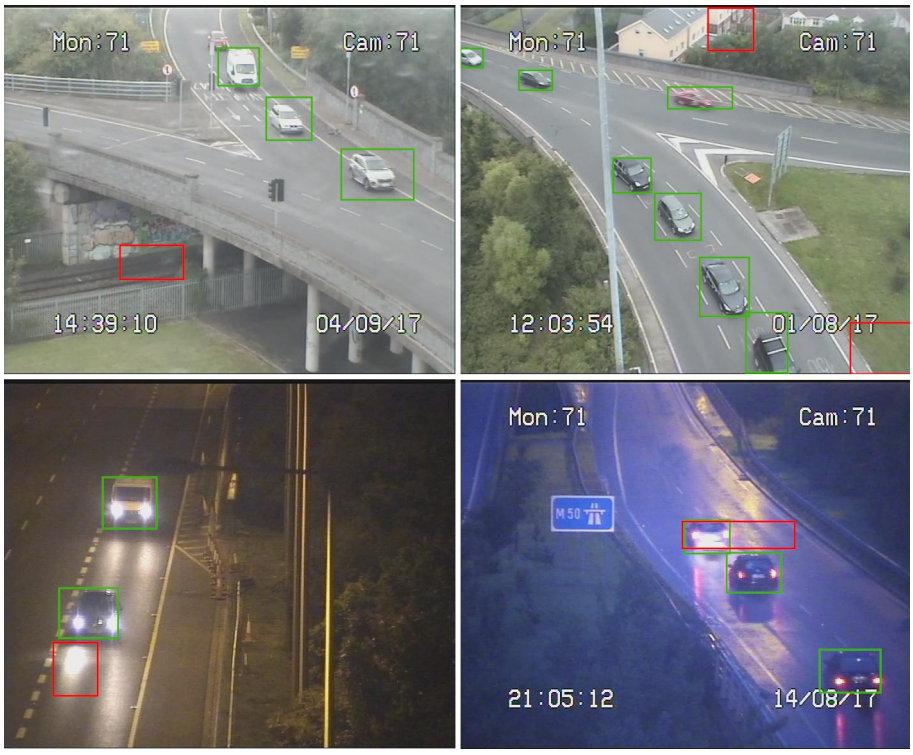

Image 5: Sample Images with predictions from our best system

TABLE OF TERMINOLOGY

Ξενόγλωσσος όρος	**Ελληνικός Όρος**
Deep Learning	Βαθιά Μάθηση
Ανίχνευση Αντικειμένων	Object Detection
Συνελικτικά Νευρωνικά Δίκτυα	Convolutional Neural Networks
Επιλεκτική Αναζήτηση	Selective Search
Συρόμενα Παράθυρα	Sliding Windows
Επεξεργασία Εικόνας	Image Processing
Ακρίβεια	Accuracy
Πλαίσια Οριοθέτησης	Bounding Windows
Παροχή Πλαισίων	Region Proposals

ABBREVIATIONS - ACRONYMS

AP	Average Precision
ILSVRC	ImageNet Large Scale Visual Recognition Challenege
IoU	Intersection over Union
mAP	Mean Average Precision
NMS	Non Maximum Suppresion
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent

REFERENCES

[1] Karen Simonyan and Andrew Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, ICLR 2015, arXiv:1409.1556 [cs.CV], 2015.

[2] Deng, J. and Dong, W. and Socher, R. and Li, L.-J. and Li, K. and Fei-Fei, L., *ImageNet: A Large-Scale Hierarchical Image Database*, CVPR09, 2009.

[3] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei, *ImageNet Large Scale Visual Recognition Challenge*, IJCV, 2015.

[4] L. Fei-Fei, R. Fergus and P. Perona, *Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories*, IEEE. CVPR 2004, Workshop on Generative-Model Based Vision, 2004.

[5] G. Griffin and A. Holub and P. Perona, *Caltech-256 Object Category Dataset*, California Institute of Technology, 2007.

[6] Alex Krizhevsky, *Learning Multiple Layers of Features from Tiny Images*, 2009.

[7] Alex Krizhevsky and Sutskever, Ilya and Hinton, Geoffrey E, *ImageNet Classification with Deep Convolutional Neural Networks*, Advances in Neural Information Processing Systems 25, 2012, pp.1097-1105.

[8] Matthew D. Zeiler and Rob Fergus, *Visualizing and Understanding Convolutional Networks*, Dept. of Computer Science, New York University, USA, 2013.

[9] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke and Andrew Rabinovich, *Going Deeper with Convolutions*, arXiv:1409.4842 [cs.CV], 2014.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun, *Deep Residual Learning for Image Recognition*, arXiv:1512.03385 [cs.CV], 2015.

[11] Joseph Redmon, Santosh Divvala, Ross Girshick and Ali Farhadi, *You Only Look Once: Unified*, Real-Time Object Detection, arXiv:1506.02640 [cs.CV], 2015.

[12] J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, and A.W.M. Smeulders, *Selective Search for Object Recognition*, IJCV 2012, 2012.

[13] Everingham, M. and Van~Gool, L. and Williams, C. K. I. and Winn, J. and Zisserman, A., *The PASCAL Visual Object Classes Challenge 2012 VOC2012 Results*, 2012; http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[14] Chollet, Francois and others, Keras, 2015; https://keras.io.

[15] Jonathan Krause, Michael Stark, Jia Deng and Li Fei-Fei, *3D Object Representations for Fine-Grained Categorization*, 4th IEEE Workshop on 3D Representation and Recognition, at ICCV 2013 (3dRR-13). Sydney, Australia, 2013.

[16] Diederik P. Kingma and Jimmy Ba, *Adam: A Method for Stochastic Optimization*, arXiv:1412.6980 [cs.LG], 2014.

[17] Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro and Benjamin Recht, *The Marginal Value of Adaptive Gradient Methods in Machine Learning*, arXiv:1705.08292 [stat.ML], 2017.

[18] Pedro F. Felzenszwalb and Daniel P. Huttenlocher, *Efficient Graph-Based Image Segmentation, International Journal of Computer Vision*, 2004.

[19] Redmon, Joseph and Farhadi, Ali, *YOLOv3: An Incremental Improvement*, arXiv, 2018.