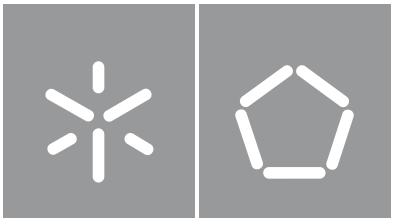


Universidade do Minho
Escola de Engenharia

Filipe Pimenta Oliveira Monteiro

Application of Semantic Segmentation through data acquired from sensors

Dissertação de Mestrado
Mestrado Integrado em Engenharia Informática

Trabalho efetuado sob a orientação do(a)
Paulo Jorge Novais

novembro de 2021

Direitos de Autor e Condições de Utilização do Trabalho por Terceiros

Acknowledgments

I would like to express my deep gratitude to my supervisor, Professor Paulo Jorge Novais, for this opportunity in accepting me and the suggested topic, and for always supporting and providing every tool needed for the development of this work.

To my whole family which was always supportive and comprehensive about my needs, providing me the best environment I could need for all these years of academic life. Their belief in me and my work kept me motivated from the day I entered this journey and will keep motivating me on the future ahead.

I can not thank enough professor Filipe Gonçalves for the guidance and tutoring throughout the entire work which was fundamental in achieving this best version of me and this work. His words of courage and incitement kept me at bay, focused on the task at hand, but never forgetting and always caring me as a friend. Without a doubt, a person I look up to, where his experience will serve as counselling for the future.

To Leonardo, for always pushing me to do better and achieve greater heights, for always being by my side and help me brainstorm difficult problems, both work related and personal. Without his guidance and his friendship I would not be as developed as today.

Lastly, but certainly not the least, I am in great debt to all my friends which made me what I am today. Without their company and support, no special memories would I have from these times, from the Hackathons we shared, the after-exam nights to the pandemic crisis we lived in. From the bottom of my heart, friends of a lifetime.

Filipe Monteiro

Statement of Integrity

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Resumo

Aplicação de Segmentação semântica a partir de dados recolhidos por sensores

Hoje a inteligência artificial é uma ferramenta com grande presença nas nossas vidas, muitas vezes sem a nossa perceção. De utilizações simples como assistentes pessoais - *Alexa* e *Siri* -, e personalização de publicidade baseada nos nossos gostos - apresentação de publicidade relevante baseada no nosso histórico de pesquisas -, para casos mais complexos como robôs e casas, veículos e cidades autónomas. A presença da inteligência artificial cresce, e apesar de estarmos ainda bastante longe da nossa ideia de *'General AI'* - uma máquina capaz de executar qualquer função de forma autónoma -, cada dia nos encontrámos mais perto.

Na última década múltiplas aplicações da inteligência artificial realizaram avanços significativos como, por exemplo, as primeiras implementações de veículos autónomos que foram introduzidas por várias empresas, sendo a mais reconhecida a *Tesla*. Para chegar a este patamar nesta revolução da inteligência artificial foi necessário uma grande quantidade de estudo e descobertas onde, entre elas, duas se destacam com grande importância: Deteção de Objectos e Segmentação Semântica, ambas relacionadas. Estas são responsáveis por recolher conhecimento sobre o ambiente em que se encontra a máquina de forma a esta executar ações. Apesar de estarem ambas conectadas, a última pode ser considerada uma melhoria perante a primeira em relação á sensibilidade do erro associado a cada entidade detetada, assim como a nova capacidade de detetar todos os objetos e os seus respectivos tipos. Para isso as máquinas necessitam cada vez mais de dados para analisar e treinar, obtidas de vários possíveis sensores como radares, diferentes tipos de câmaras, LiDAR, entre outros.

Esta dissertação tem então o objetivo de estudar o uso de técnicas de Segmentação Semântica e as suas aplicações na categorização de imagens provenientes de sensores do tipo câmara para explicar as suas descobertas e desafios, assim como melhorar e ultrapassar obstáculos existentes. Os dados consistem de *scans* das estradas, capturadas do POV de um veículo (*KITTI360*, sendo facilmente adaptada a utilizar *scans* de outros contextos para categorizar imagens mais "comuns" (*COCO*). Atualmente os trabalhos do *DeepLab* (com o desenvolvimento do modelo *deeplabv3*[1]) conseguiram resultados com bastante sucesso, ultrapassando desafios anteriormente existentes tal como o tratamento dos limites de cada entidade da imagem, com vizinhanças bem definidas, sendo ao mesmo tempo capaz de ser executada em máquinas com poucos recursos, sendo por isso o ponto de início para este trabalho.

Palavras-chave: Segmentação Semântica de imagens, veículos autónomos, redes neuronais convolucionais, segurança automóvel

Abstract

Application of Semantic Segmentation through data acquired from sensors

Today, AI is very important in our lives as its used all around us without our knowledge. From simple things such as personal assistants like *Alexa* and *Siri*, and advertising algorithms focusing on our tastes - *Netflix* on the recommendation of movies or, even more common, the presentation of advertising based on our search history -, to robots and to smart houses, cities or even vehicles. The presence of AI is increasing and even if we are still far away from our *'General AI'* ideology, a machine capable of anything autonomously, each day we get closer.

In the last decade multiple applications of AI have been through breakthroughs. For example, the first implementations of autonomous vehicles were introduced by *Tesla* and other companies. A number of discoveries must have been made to achieve this revolution of AI performance and, among them, is two of the most important developments: Object Detection and Semantic Segmentation, closely related to each other. These are responsible for understanding the environment so the machine can take actions, being the latter an improvement of the first in terms of sensibility error associated to each entity detected as well as being able to detect its corresponding type, in a pixel level. These machines require more and more data to analyse, having many types of sensors in order to collect information, such as radars, cameras, LiDAR, among others.

This work falls in the study of the use of Semantic Segmentation techniques and its application on categorising data from image related sensors in order to explain its breakthroughs and challenges, as well as improving and overcoming such obstacles. Data will consist mainly of scans from outdoor/self-driving cars POV (*KITTI360*) with the ability to be used with other types of data such as indoor scans (*COCO*), to explain both road and more day-to-day images semantic compositions, applied on a state-of-art solution. Consecutively we will perform a process of optimisation in order to reduce computation costs. Currently the works of *DeepLab* (with the research of *deeplabv3*[1]) have achieved a high success on Semantic Segmentation overcoming previous problems such as handling component boundaries with more refined lines while keeping it fairly easy to run on more less powerful machines, being the start point for this work.

Keywords: Image Segmentation, Semantic Segmentation, Deep Learning, Self-Driving, Automotive Security

Contents

List of Acronyms

ASPP Atrous Spatial Pyramid Pooling. 23, 50

CNN Convolutional Neural Networks. 10, 11

CRF Conditional Random Field. ix, 17, 18, 23, 34, 50

HOG Histogram of Oriented Gradients. 8

IoU Intersection over Union. 22

LiDAR Light Detection and Ranging. iv, v, ix, x, 1, 2, 6, 7, 13, 21, 22, 24, 25, 50, 51

mIoU Mean Intersection over Union (Jaccard Index). 22, 29, 32, 50, 51

NIR Near-Infrared. 1

POV Point-of-View. iv, v, 22

RGB-D RGB with Depth perception. 1, 13

SIFT Scale-Invariant Feature Transform. 8

List of Figures

List of Tables

Chapter 1

Introduction

The detection of objects and entities on images or other sensor's data is a relatively simple task for a human brain, from an early age. However for a computer it is a very hard action. From a decade ago multiple techniques appeared in order to solve and optimise this action, with ones like Object Detection and Semantic Segmentation being among the most popular ones. They differ on their on the architecture used as well as the application and use cases available, with autonomous driving, being the focus of the work, the most applicable when talking about Semantic Segmentation.

Image semantic segmentation is a field for image processing where the main goal is to achieve full and accurate separation of entities on an image. Initially these were achieved by using multiple techniques such as threshold segmentation, region segmentation, edge segmentation, texture features and clustering, among others[2]. Deep learning then started to gain some ground work leading to an explosion of scientific works about applying deep neural networks on this segmentation task. Having a better performance, new tasks where attempted such as image instance segmentation, where each different object (even from the same class) would be considered unique and separated.

Applications ranged from detecting bad cells at microscopic level[3], to identifying aerial ground images on the type of ground[4]. Currently, this technique has been more focused on the self-driving context for autonomous vehicles, which will serve as the start point for our work.

1.1 Motivation

When we think of a future where AI is implemented all around us, one of the main attractions we might think of is self-driving vehicles. These will most likely use a combination of multiple sensors from RGB-D/NIR cameras to LiDAR or similar sensors in order to collect information of their environment to base their decisions on. While today there are already algorithms using only cameras for this purpose, they still have flaws from misclassification (which in this context can be pretty harmful) to require powerful resources in order to use them. On the other hand, LiDAR sensors can be more precise but costing many times more than a simple camera sensor.

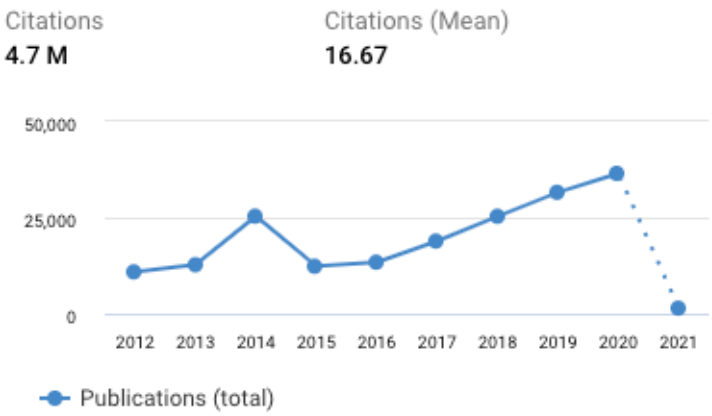


Figure 1: Popularity growth for research papers focused on semantic segmentation.[1]

Since 2012 this has been an area of interest for investigation (as shown in Figure 1), with a continuous growth in the number of papers boarding it, due to the possibilities and use-cases available for this type of technology. Recent studies have been more focused on self-driving cars, resulting in a higher number of datasets developed for this purpose, but not only. As cameras are not the only sensors that can be used to retrieve environment data, LiDAR sensors have been another common type which, based on 3D points of the inferenced area, can provide decent results for it's segmentation. This technology is highly used up to day, for example, for segmenting aerial LiDAR data to aid on agriculture[5] and similar tasks or even aid robots to achieve multiple goals, such as navigation and task execution[6]. There is also a current focus among the scientific community on discussing its value facing camera sensors, based on their cost to performance aspect (LiDAR are notably more expensive than cameras).

Being images originated from a camera type sensor much easier to manipulate and explore, these will become the object of focus for this work. The fact that many companies and organisations to this day use images as an input for Semantic Segmentation also increases the chance of finding better datasets, as well as justify the need to explore this technique and what it can do to aid us in the constant evolution of technology.

These types of techniques usually require lots of computation resources, as well as big amount of data to train these. Different optimisations were developed and studied while applied on an Deep Learning context but only on a high level of research domain. Combining these two will provide us an insight of what we can do with these algorithms, how these datasets are built and used, as well as provide greater insight on optimisation of deep learning algorithms, a topic usually not very friendly and academic.

[1] Source: App Dimensions

1.2 Goals and Contributions

The main reasoning of this work is to explore the performance and reliability of semantic segmentation using state-of-art works inserted on the context of self-driving vehicles, using new and recent datasets for testing. Throughout this project we expect to evaluate the full reliability of these algorithms in the real-world, as well as attempt to improve their results with added data.

Also related to the improvement of the usability of these algorithms, we will explore and gather insights on the topic of deep neural networks compression, in which we intend to show the main techniques available in the research domain, as well as apply them and report our findings on their pros and cons using different Evaluation metrics and inferenced images as proof. This way this research can provide insights for others starting their journey on these high-level optimisations techniques.

Finally, this work will serve as a proof of concept with the potential of what state of the art techniques can achieve, when faced in new environments, as well as demonstrate the flow of work and challenges that need to be surpassed taking into account real world scenarios.

1.3 Methodology

Task	**Nov**	**Dec**	**Jan**	**Feb**	**Mar**	**Apr**	**May**	**Jun**
State of Art Survey	X	X	X					
Acquisition and Preparation of datasets	X	X	X					
Application of Semantic Segmentation algorithms			X	X				
Benchmarking and Evaluation of performance			X	X				
Identification of incorrect classifications			X	X				
Improvement of classification performance			X	X	X	X		
Optimisation of computational costs				X	X	X	X	
Dissertation writing	X	X	X	X	X	X	X	X

Table 1: Work schedule.

This research will start by diving in the theme of Semantic Segmentation and their related techniques, getting to know how these appeared, in which contexts they were used and how they functioned. Following this analysis we will turn our focus on the main technique to be used and explore all the state-of-art architectures available and their unique features, ending up with an architecture to be used as a starting point for the work.

With the algorithm explored, we will search and analyse all the different alternatives of the available open-source datasets to prepare on our algorithm, checking benchmarks and common techniques. With the dataset chosen, we will prepare it for training and evaluation purposes.

Finally, we will benchmark the base architecture on the chosen dataset, and proceed with a research of how to improve it, using all the backbones and available options of the chosen model in order to expand

the range of conclusions.

Based on these results, the research will follow to approach the model with the intention of reducing it's computational resources while keeping its segmentation performance, concluding how much can it be improved without penalising its output.

1.4 Document Structure

This work will consist of several chapters with the following composition:

- State of the Art (2): here will be introduced the concept of semantic segmentation, its advantages and disadvantages, explaining as well the basic functioning of this technique. Later in this chapter, we will also overview the most impactful algorithms created for this technique, discussing their unique features and achievements;
- Preliminary Studies and Approach (3): this chapter will consist of the starting point and initial assumptions made by us, while explaining how the work will proceed as well. We will also display some initial results found;
- Results and Findings (4): analysis and explanation of the different results based on the proposed algorithms and datasets, as well as exploration of optimisations directed to deep neural networks, applied to our algorithms;
- Conclusion (5): a conclusion relative to the overall work so far done, the problems and limitations we encountered, and what we intend to achieve in the near future.

Chapter 2

State of the Art

As of today AI has an enormous mark on our everyday life thanks to the increase study and its application, as well as the increase on the resources of available hardware. But with this extra computational power and complexity of models comes the need for more data, leading to the creation of *Data Warehouses* in order to store all unnecessary data. This data is information shared from people online, companies, sensors from cities and vehicles among others, where its collection and processing is done in an autonomous way. Focusing on this work main target, the research of semantic segmentation techniques, data prepared by users based on images are being used in order to develop software capable of creating semi-autonomous virtual filters to apply on images, for instance, locating specific objects or even, utilised by big e-commerce companies, for searching for specific objects on a store with similar characteristics[7].

In the automotive world, the use of AI to achieve levels of automation only started to be achievable when new, improved and cheaper camera sensors, radars and computers appeared. Until then, it could only be considered experimentation. In the last decade this improvement was noticeable, with recent years having the first implementations of autonomous driving using semantic segmentation techniques or similar, like object detection (shown in Figure 2), constantly improving with the added data and optimisation of these algorithms. One major example of this is *Tesla* which actively captures user driving data for active training of their algorithms[8].

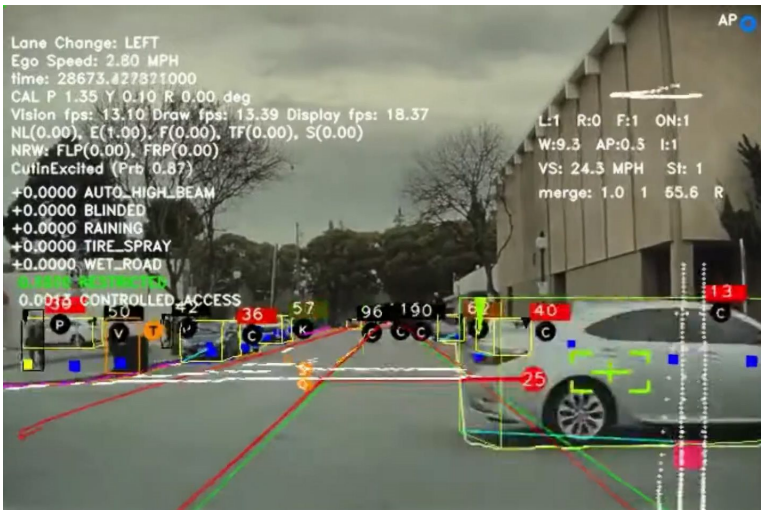


Figure 2: How the Tesla algorithm sees the world.[1]

2.1 Related Work

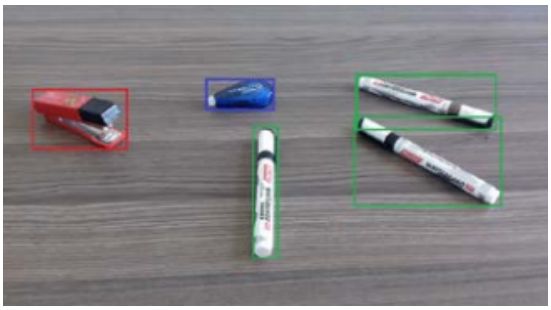
This type of task, Semantic Segmentation, is closely related to other techniques such as Instance Segmentation, where its only difference is the extra information the latter provides. While Semantic Segmentation computes each pixel class, Instance Segmentation goes further and distinguishes the pixels from the same class that belong to different entities. This is achieved by a combination of Object Detection, an algorithm to encounter objects in an image using bounding boxes with its classification, and Semantic Segmentation, to properly classify each pixel class.

As mentioned earlier, Object detection is another type of a similar classification technique, where it uses bounding boxes to delineate the entity on the image and classify it. Famous algorithms include *YOLO*[9], *RetinaNet*[10], *R-CNN*[11] and others[12]. The comparison between all these techniques can be seen on Figure 3.

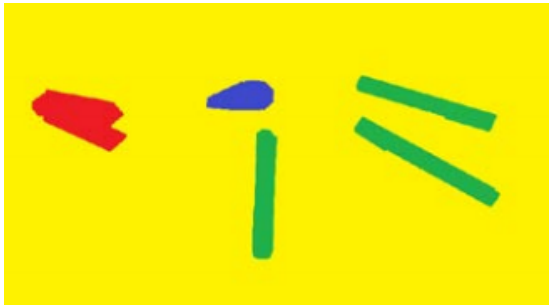
Despite the fact image based techniques are more experimented given the easy availability and use of image based sensors such cameras, there is also another important sensor used specially on autonomous vehicles: the LiDAR. This sensor involves shooting light rays through an environment and receiving the bounced lights on the sensor, generating a 3D [14] representation of the world based on a group of Point Cloud (see Figure 4). In the past years the discussion about LiDAR vs camera sensors to be used on autonomous vehicles has been an hot topic regarding their performance versus cost ratio[15], but more and more investigators agree that cameras should be the future for autonomous driving. Given the fact LiDAR investigation is complex, high demanding in resources and high time consumption, this won't be researched, although it is worth mentioning that more of the successful techniques created are based on the transformation of these 3D point-clouds into 2D images for a network to segment, such as *SPLATNet*[16], *RangeNet++*[17], among others[18][19][20]. In similar fashion, some algorithms such as

[1]Source: https://www.whichcar.com.au/car-advice/this-is-what-tesla-autopilot-sees

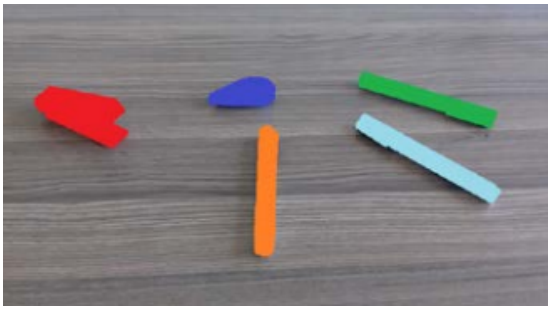
(a) Input image.



(b) Object Detection.



(c) Semantic Segmentation.



(d) Instance Segmentation.

Figure 3: The main techniques of image classification[13]

RoIFusion[21] use this type of information to calculate 3D bounding boxes around the entities surrounding it with the aid of image data as well.

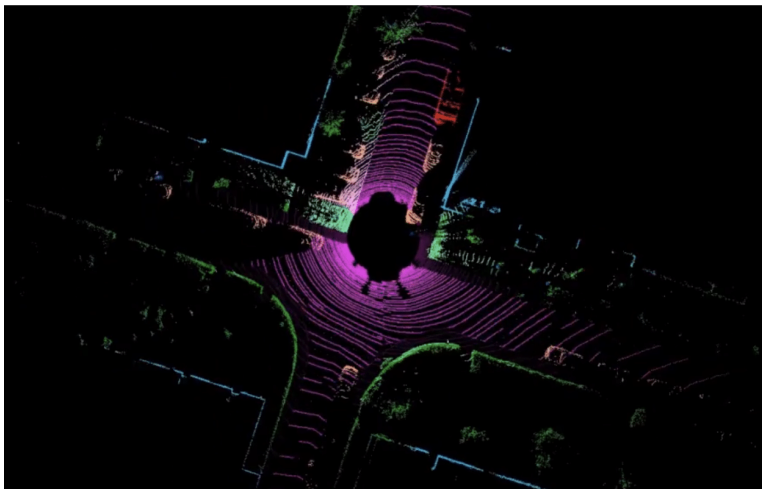


Figure 4: Example of LiDAR data.[2]

[2]Source: https://blogs.nvidia.com/blog/2020/03/11/drive-labs-multi-view-lidarnet-self-driving-cars/

2.1.1 Object Detection

Object detection consists on a technique that detects where and which objects are on an image by providing the bounding boxes of each one. Today this technique is based on deep convolutional neural networks, but previously were built based on hand-crafted feature extractors, e.g. Histogram of Oriented Gradients [22], Scale-invariant feature transform [23], and many others. Their evolution timeline can be seen on Figure 5.

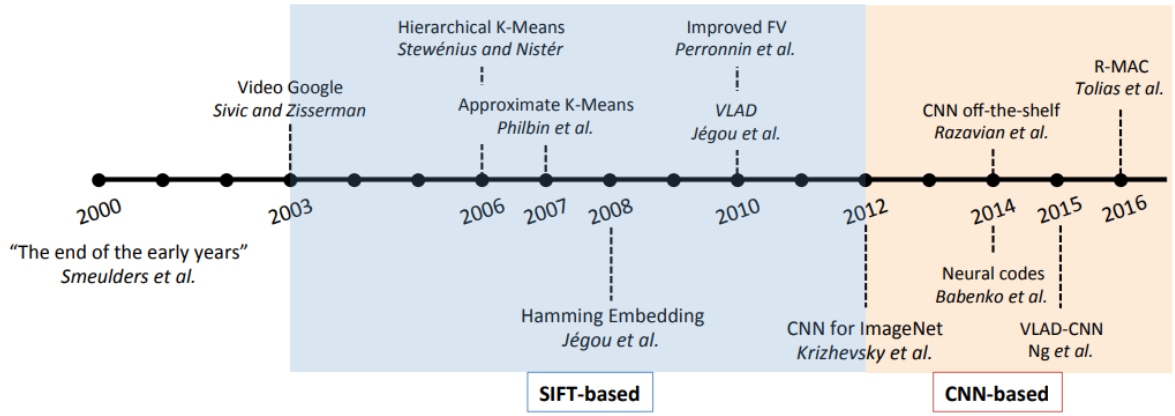


Figure 5: Timeline of usage for both *SIFT* and *CNN* methods for object detection [24]

2.1.1.1 HOG and SIFT

HOG and SIFT, consist on the computation of the distribution of intensity gradients or edge directions. The computation is executed on small patches of the input image, similar to applying a convolutional filter, resulting on, for each pixel, an histogram of gradient directions (see Figure 6 and 7 for a visual interpretation). This method suffers on the performance, as calculation an histogram for each pixel can be costly the bigger the resolution of the image, as well as it works best where objects do not vary much in shape, as colour is not directly used and similar objects may be assumed to be of the same type. Also, simple transformations to the image such as rotation can drastically reduce accuracy. These two feature extractors vary on the approach of dividing and applying the computation. For instance, SIFT divides the images into smaller patches consisted of HOG computations.

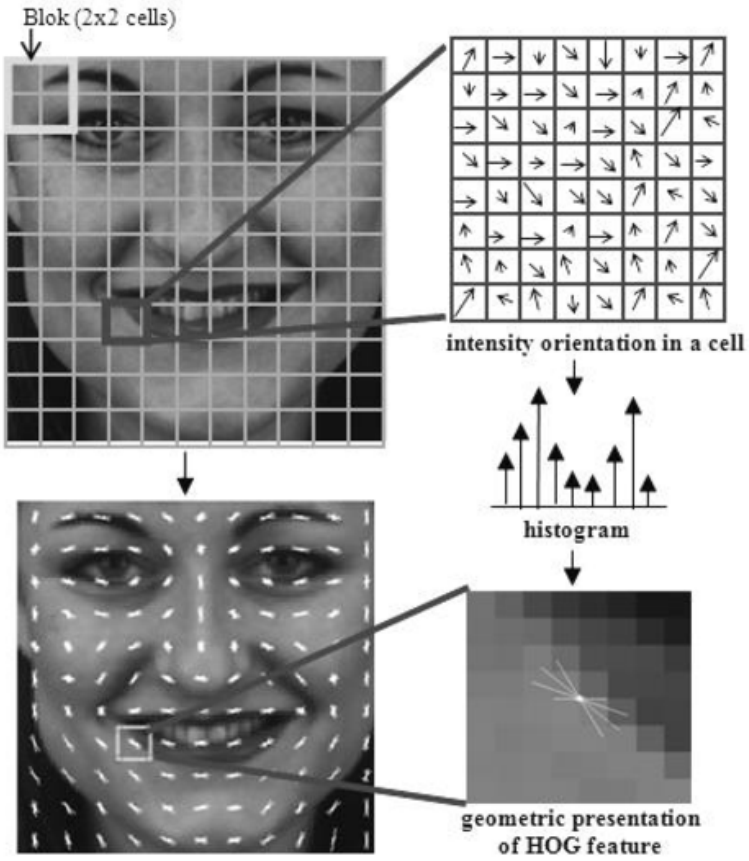


Figure 6: Histogram of Oriented Gradient feature extraction example.[3]

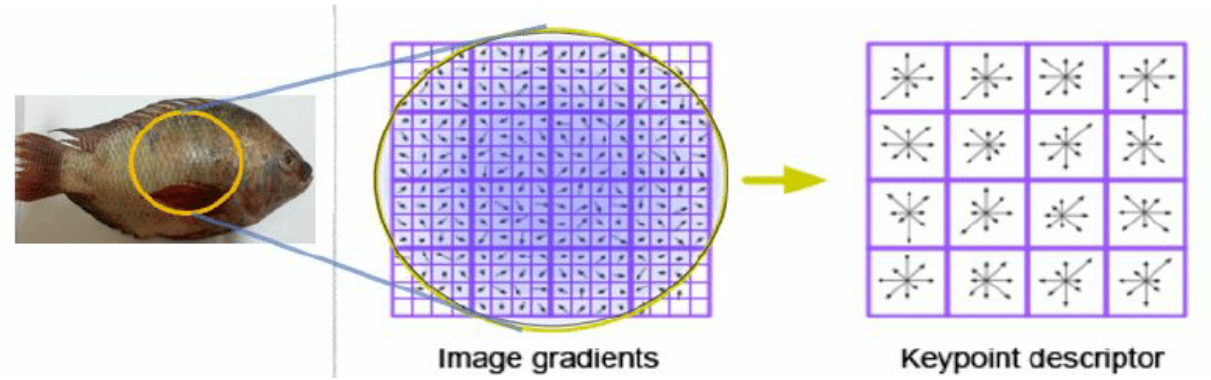


Figure 7: Scale-invariant feature extraction example [25].

[3]Source: https://heartbeat.comet.ml/introduction-to-basic-object-detection-algorithms-b77295a95a63

2.1.1.2 One and Two Stage Detector

When CNN started to be experimented with, object detection benefited as the previous features extractors manually built could now be automated and trained using these new convolutional layers, specially built for the task. These CNN methods can be separated onto two categories: two stage (Figure 8) and one stage (Figure 11) detectors, being the main difference that the first makes use of a second parallel processing algorithm's result (second stage) to combine with the first stage in order to provide a prediction.

As mentioned, Two stage detectors make use of two different algorithms for two distinct tasks: one model serves the purpose of detecting the region of interest of the object, where its output is used on another model responsible for classifying and refining the localisation of the object. This approach, although very powerful and accurate, comes at a high computational cost due to using two different models for the final prediction, while also having a higher inference time compared to one stage detectors.

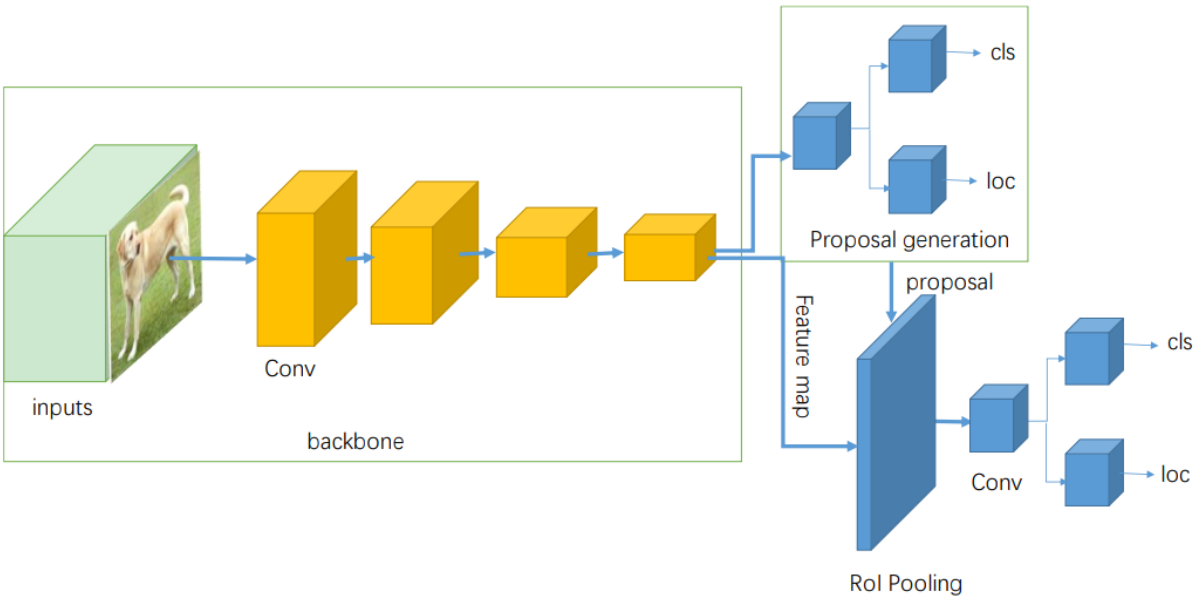


Figure 8: Example of a two stage architecture.[12].

One of the first examples of a two stage detector was *R-CNN*[26]. As mentioned, the algorithm makes a classification based on two steps: (1) extracting the regions of interest; (2) classifying each region of interest (see Figure 9).

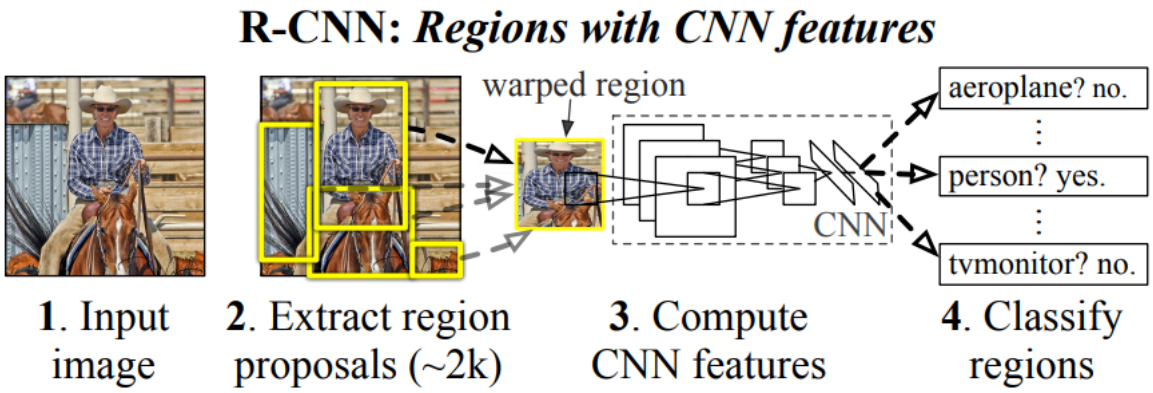


Figure 9: Overview of R-CNN pipeline.[26]

The first stage, for the collection of regions of interest uses *selective search*, an algorithm based on computing hierarchical grouping of similar regions based on colour, texture, size and shape compatibility. This algorithm consists on the segmentation of pixels based on their intensity using a graph-based segmentation method by Felzenszwalb and Huttenlocher [27], with the objective of reaching a point where every single region cannot be sub-segmented any further (see Figure 10 for visual understanding), resulting on a over-segmented image. Although we are technically segmenting the image, these segmentations are not accurate as most of the objects get over-segmented (i.e. color changes on a single object can create multiple segmentation masks), consisting then of multiple sub-segmentations. The final stage applies a CNN on each cropped part of the image containing a main region of (assuming this main region corresponds to an object) and classifies it.

Figure 10: Example of the selective search algorithm.[4]

On the other hand, one stage detectors consist of an architecture where the image is only processed once, resulting on bounding boxes of the objects and respective classes, with low computational cost and

[4]Source: Stanford Presentation

inference time, but where the accuracy and resulting bounding boxes are not as refined as from a two stage detector.

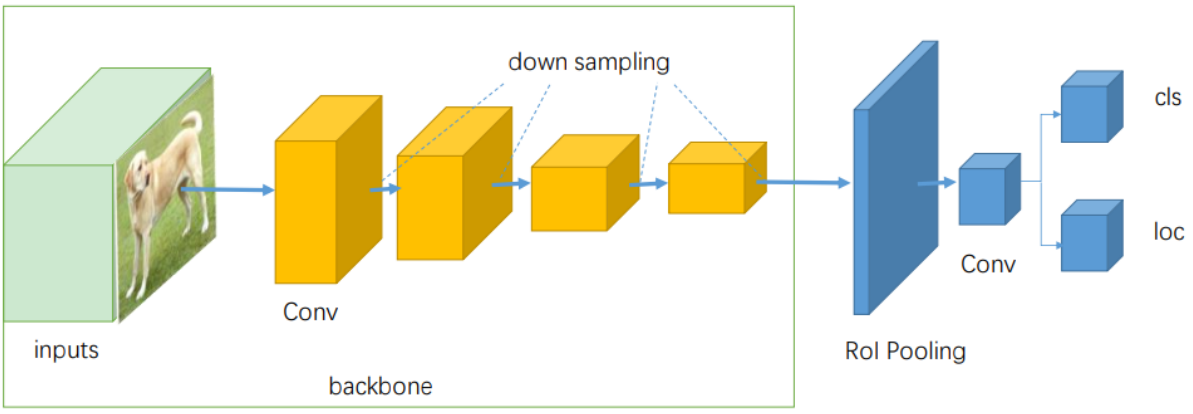


Figure 11: Example of a one stage architecture.[12].

YOLO[9] is one of the most well known architectures when we think about Object Detection, firstly published in 2015. Improved over the years [28][29], the model uses a single neural network (see Figure 12) to calculate both the location of the bounding boxes and its class, by dividing the image in a SxS grid (hyper-parameter) and detecting the object inside each cell.

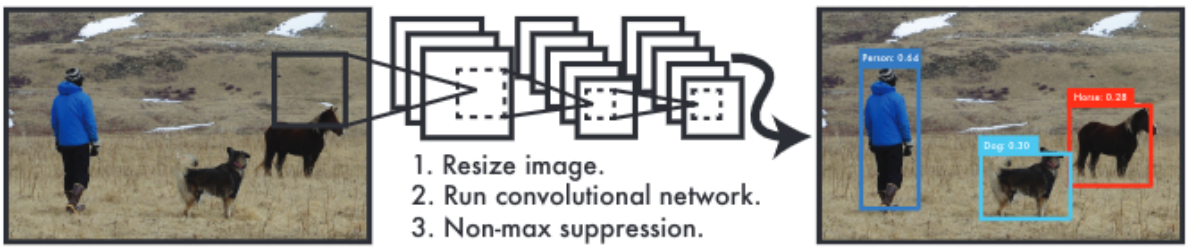


Figure 12: Overall pipeline from *YOLO*[9] model.

Analysing Figure 13 we can see how the model divides the image in a grid, predicting bounding boxes with x, y, w, h, confidence and class probabilities for each grid cell. The main key difference is that they limit the way the model detects the objects and prevent multiple detections of the same object, by keeping spatial constraints on the grid cells. Further improvements managed to mitigate problems such as the lack of ability to detect correctly groups and smaller objects, as well as reduce the localisation errors available on the first iteration, by applying random multi-scale training during the training, newly organised convolutions on the network, among others.

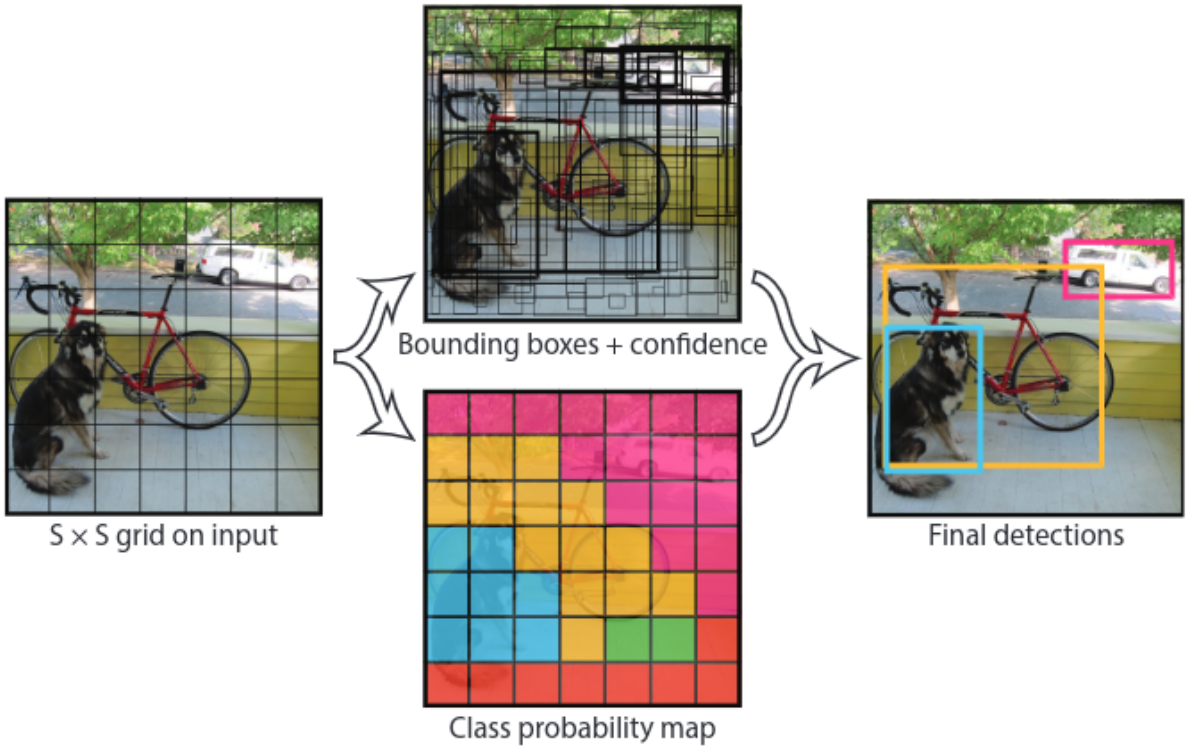


Figure 13: Prediction method used by the *YOLO* [9] model.

As stated previously, we decided to study not only how image sensor data is used, but also LiDAR data, although on a much smaller scale. As discovered, Object Detection can also be applied using two distinct pipelines: (1) make use of LiDAR and image data creating an hybrid architecture [21]; (2) make use of image data only, based on a RGB-D sensor where depth information is available [30]. These case studies were not on the focus as LiDAR is technically too demanding for this type of work.

2.1.2 Semantic Segmentation

Since around 2012 works related to image segmentation using deep neural networks have been appearing, but in 2015 there was a special work called *Unet*[31] that started to bring more and more attention to this technique. This work was created with the intent of showing how to successfully train a deep network with a small training dataset, strongly relying on data augmentation techniques. It also showed how powerful the network created was on semantic segmentation due to it's configuration - a contracting and expansive path (left and right side of the network, respectively) resembling a encoder-decoder architecture, with added filters from the left side on the right side of the network (skip-connections) - check Figure 14 to better understand. This choice allowed for the upsampling part of the network to lose as little detail information as possible while increasing the accuracy of the pixel classification. While this technique was very successful on the biomedical area where it was inserted - identifying cells at microscope level -, even surpassing it's previous attempts, it lacked the ability to be trained and used in other contexts such as road image segmentation with high accuracy, due to its lack of depth on the network and poor optimisation on the skip-connections techniques[32].

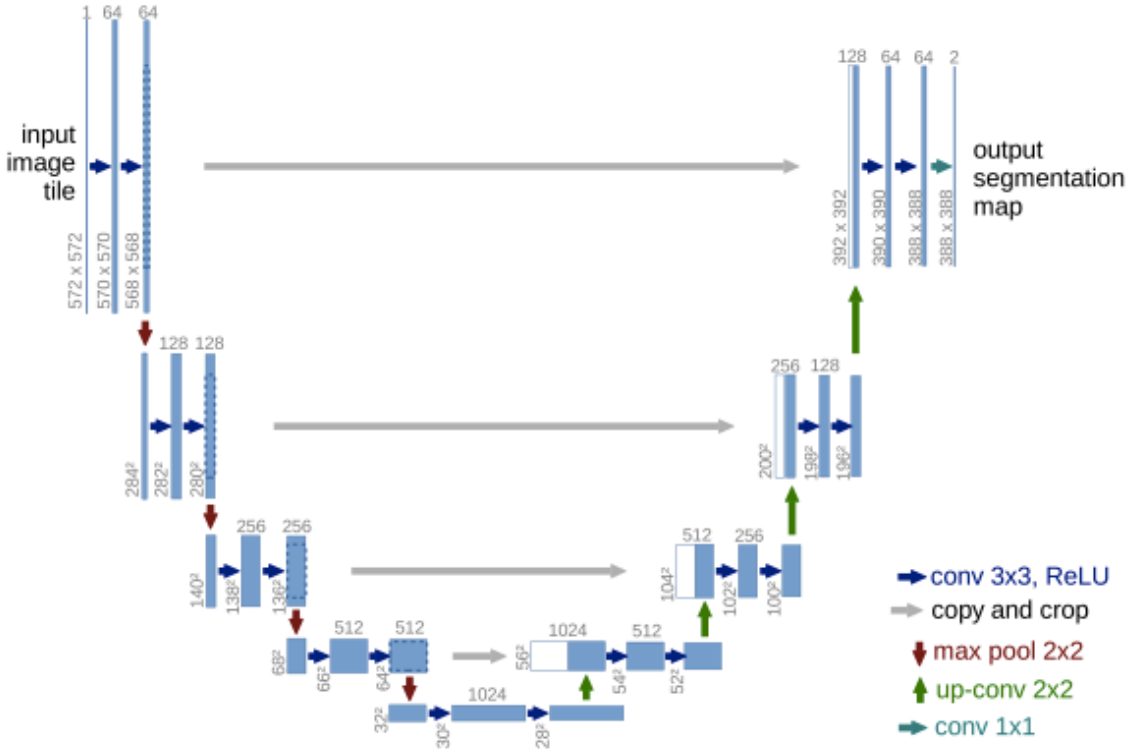


Figure 14: *Unet* architecture[31].

In the following years (2015-2018), a paper introducing a network named *DeepLab*[1] was proposed, which served as a very promising Semantic Segmentation algorithm, having multiple improvements on the overall performance on following iterations and becoming a state of the art method able to achieve amazing results without becoming too resource hungry for processing the data. It mainly consisted of a

deep convolution neural network with different *backbones* belonging to famous state of art models, with added properties in some layers (shown in Figure 15), as well as added techniques at the end of the architecture, such as:

- Atrous Convolution;
- Atrous Spatial Pyramid Pooling;
- Fully-Connected Conditional Random Fields.

Based on this architecture, we will do a deep analysis of each type of layer implemented, their computational complexity and how there can improve overall semantic segmentation results.

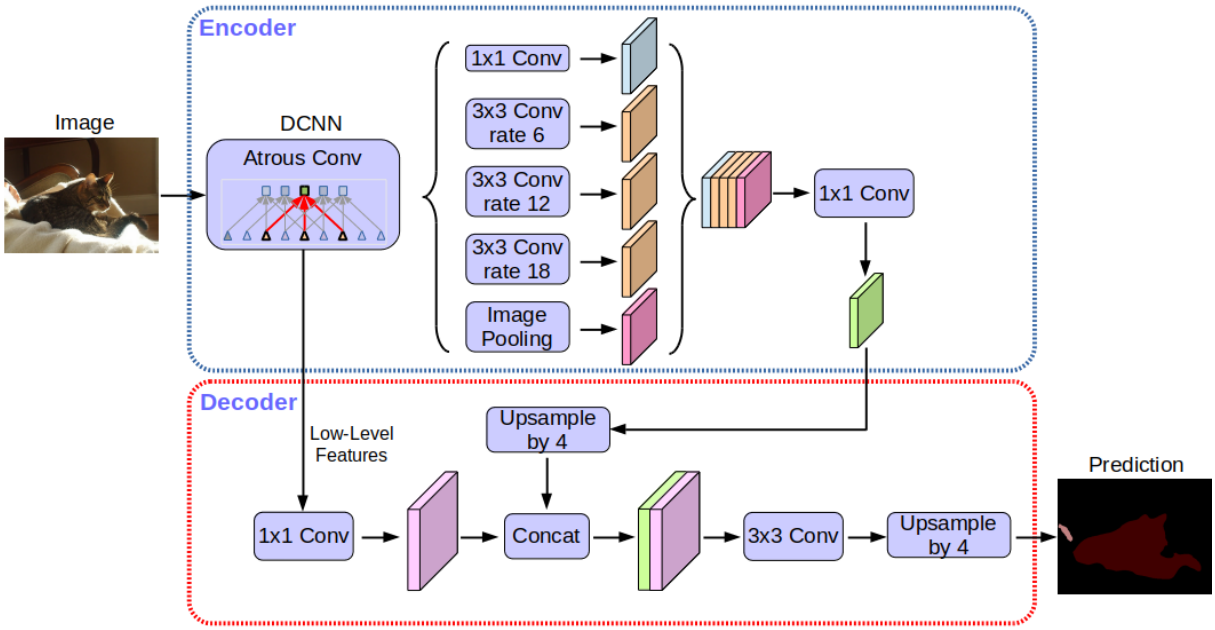


Figure 15: DeepLabV3 Architecture.
(Source: Google AI Blog)

2.1.2.1 Atrous Convolutions

Atrous Convolutions, also known as Dilated Convolutions, are a type of convolution used to create filter maps with less computational variables without loosing too much information, allowing the creation of bigger filter maps and increasing the data filtered, acting as a sort of field-of-view augmentation (as shown in Figure 16).

This technique is specially useful for image segmentation because it provides more accurate localisation of each pixel with a reduction of computation, increasing its classification performance. On the other hand, as said earlier, this skips some input data, leading to some detail loss, which has to be addressed at the end of the segmentation. The fact the algorithm also preserves the size of the original image for each layer helps remove the computation cost of upsampling the output mask (shown in Figure 17).

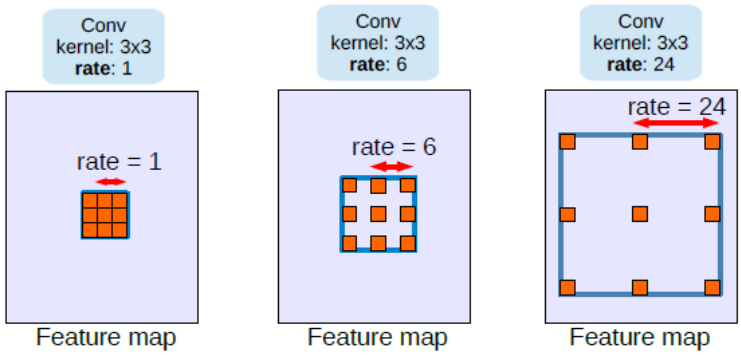


Figure 16: Base of an Atrous Convolutions[33].

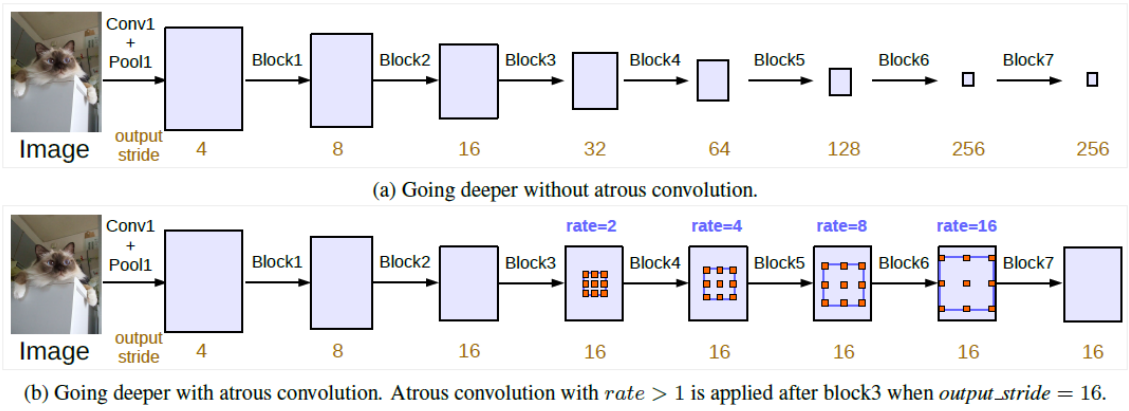


Figure 17: Use case of Atrous Convolutions vs Regular Convolutions[33].

2.1.2.2 Atrous Spatial Pyramid Pooling

As the models for semantic segmentation will encounter multiple objects of various scales, *DeepLab* decided to implement features that would expect and efficiently process these variations. To do that the group proposed two methods similar in it's core: (1) fusing the output from multiple parallel models with different output strides (corresponding to different scales of the objects of the image) and taking the highest pixel value between the multiple models. This, however, caused a big computational increase; (2) for each pixel, calculate multiple parallel filters with different rates for the convolution instead, fusing them together on further separated branches of the network (shown in Figure 18).

Based on this research, it was discovered that while the sampling rate becomes larger, the number of valid filter weights (i.e., the weights that are applied to the valid feature region, instead of padded zeros) becomes smaller, leading to more efficient pooling computations.

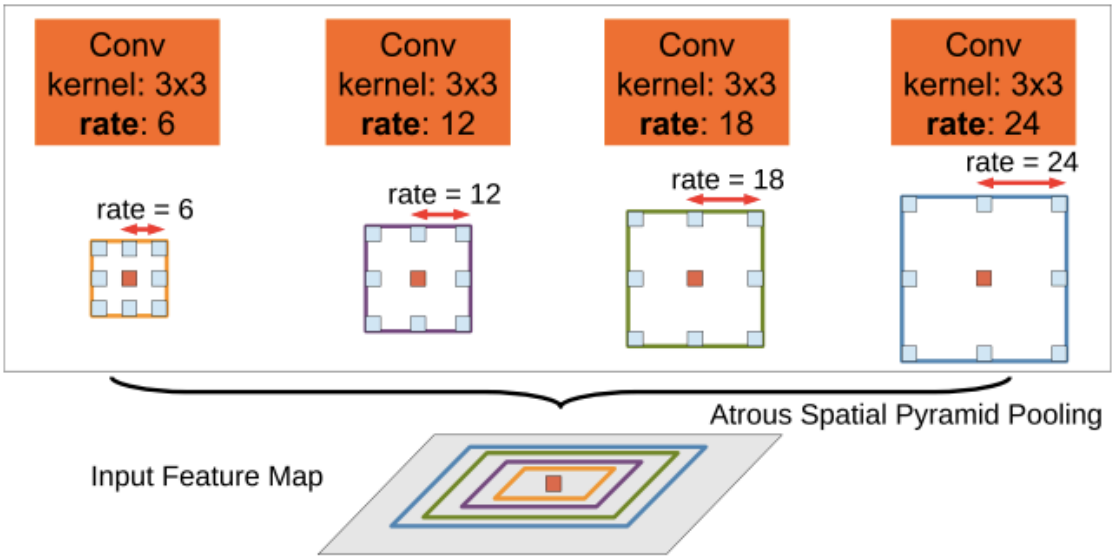


Figure 18: Computation to classify a single pixel using multiple parallel filters of different rates[1].

2.1.2.3 Fully-Connected Conditional Random Fields

One of the majors bottlenecks with deep convolution neural networks is that the networks complexity affects its ability to define fine grained boundaries between the objects on the segmentation. As such, different methods were tested and, among them, Conditional Random Fields presented to be one of the more promising.

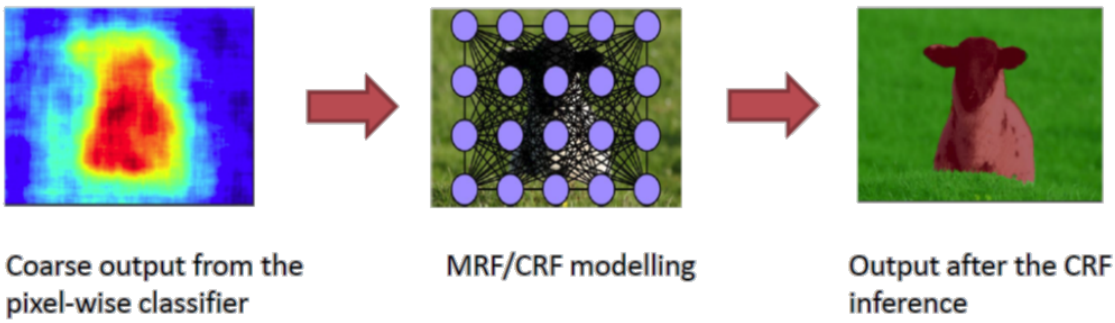


Figure 19: Difference of output without/with an CRF model[34]

This discriminative technique, is based on the concept of Markov Random Field which tries to explain the dependence or independence between random variables (as shown in Figure 19). In other words, a *Fully-Connected Conditional Random Field* is a model based on the CRF technique, allowing to calculate the likelihood of, for each pixel, neighbour pixels which should belong to the same class.

$$E\left(x\right) = \sum_{i} \theta_i(x_i) + \sum_{ij} \theta_{ij}(x_i, x_j) \tag{1}$$

$$\theta_{ij}\left(x_i, x_j\right) = \mu\left(x_i, x_j\right) \left[w_1 \exp\left(-\frac{\|\rho_i - \rho_j\|^2}{2\sigma_\alpha^2} - \frac{\|I_i - I_j\|^2}{2\sigma_\beta^2} \right) + w_2 \exp\left(-\frac{\|\rho_i - \rho_j\|^2}{2\sigma_\gamma^2} \right) \right] \tag{2}$$

Based on the research described on [1], this concept regards the following mathematical equations:

- (1): equation where x is the label associated to the pixel and $\theta_i(x_i) = -\log(P(x_i)$, having the objective of being optimised to better predict probabilities for each pixel;

- (2): auxiliary function where $P(x_i)$ the label assignment probability at pixel i, computed by the main model.

This model is then trained in order to optimise this objective result. In previous iterations of *DeepLab*, the technique improved the fine details of the resulted masks, but, with the latest *DeepLabV3* where new techniques more focused on pixel localisation (explained on the previous section) appeared, CRF became obsolete, potentially worsening the resulted masks.

Another work with exceptional results on semantic segmentation goes by the name of *Hierarchical Multi-Scale Attention* [35] which developed an improvement on the multi-scale feature collection, mentioned on the previous subsection - Atrous Spatial Pyramid Pooling. The work proposes an attention-span based model, where between each pair of scale predictions, the model learns to search the fine details aiding on the final segmentation, having this *attention* as an auxiliary mask. Not only the *attention mask* is used for segmenting on its scale, but will also aid on computing the next adjacent scale's *attention mask*.

This technique is incredibly useful because the smaller the scale of the image, the more context the filters will capture, leading to better segmentation of fine details. As can be seen in Figure 20, in the first row the thin posts are inconsistently segmented in the scaled down (0.5x) image, but better predicted in the scaled-up (2.0x) image. In the second row, the large road / divider region is better segmented at lower resolution (0.5x).

But instead of using simple methods like *max-pooling* for calculating the final classification of each pixel, the group uses these *attention masks* to calculate each scales prediction and joining these in a final prediction. This comes at the cost of having added layers for each extra scale increasing the computation cost, however, the approach proposed an hierarchical way of predicting these *masks* efficiently, reducing their computation cost.

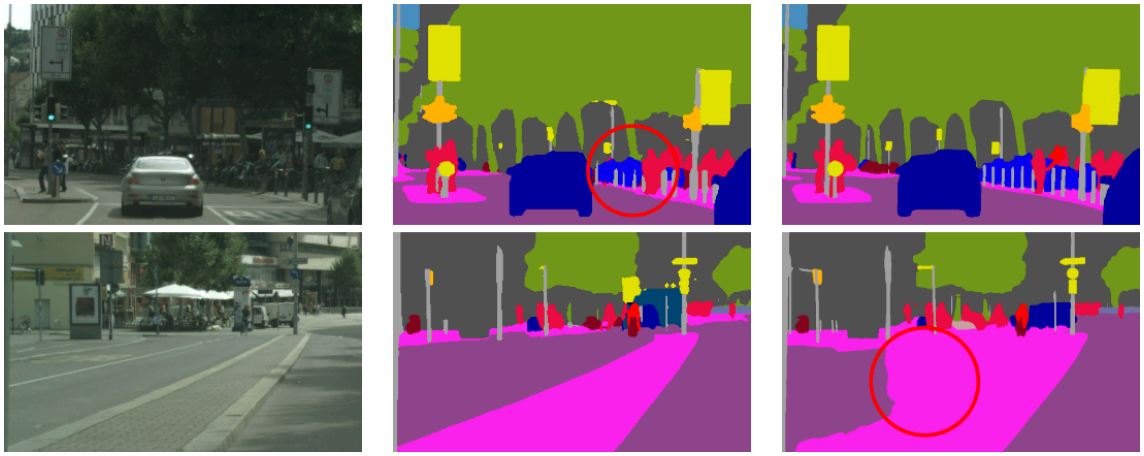
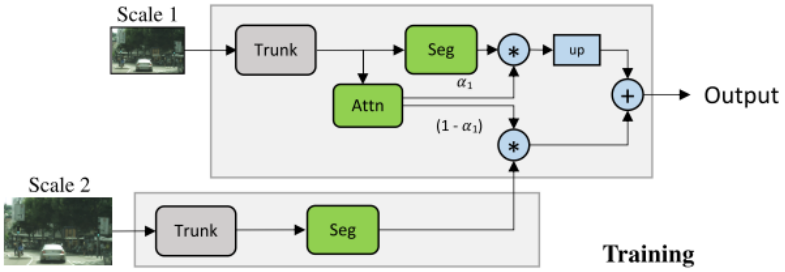
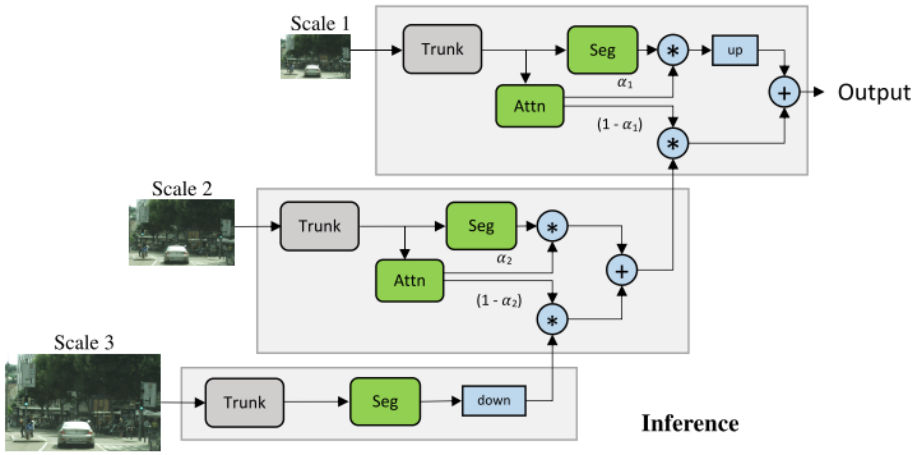


Figure 20: Illustration of common failures modes for semantic segmentation as they relate to inference scale[35].



(a) Training Architecture from Hierarchical MSA.

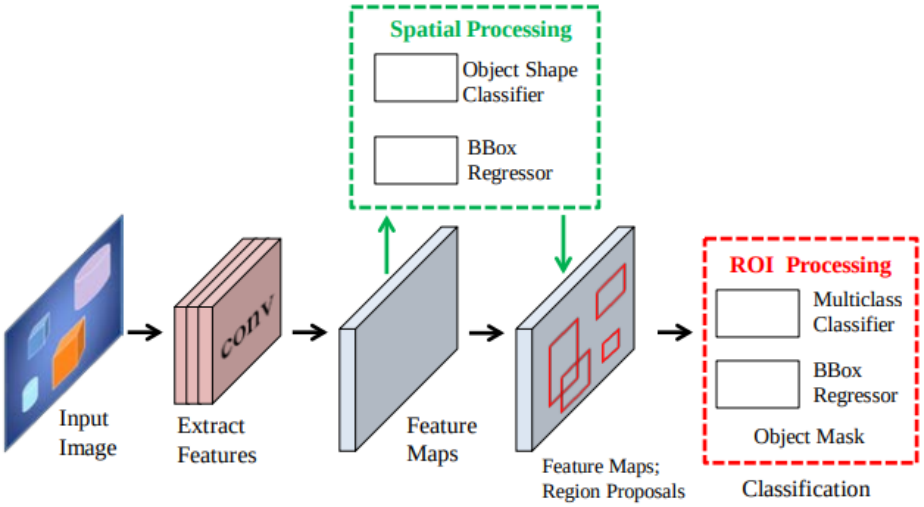


(b) Inference Architecture from Hierarchical MSA.

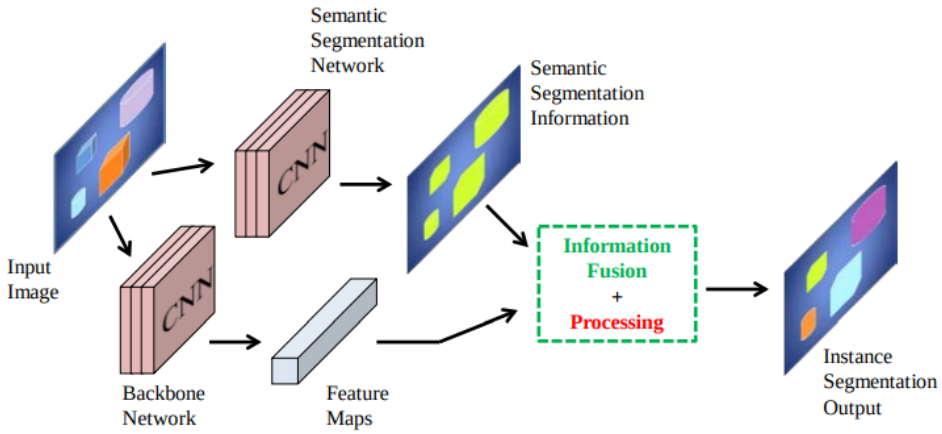
Figure 21: On both figures *Trunk* corresponds to a standard Convolutional Network architecture[35].

2.1.3 Instance Segmentation

Exploring Instance Segmentation we came to the conclusion that most of the approaches use a combination of Object Detection and Semantic Segmentation, meaning they either segmentate first and detect the different segmentations after or the other way around, clustering based on pixel classification, and many others (see Figure 22 for some examples).



(a) Detection followed by Segmentation Techniques approach.



(b) Labelling Pixels Followed by Clustering Techniques approach.

Figure 22: Some of the different Instance Segmentation architectures[13]

Mask R-CNN[26] is an Instance Segmentation algorithm developed based on the detection followed by segmentation approach presented on Figure 22a. This architecture is based on an improvement of the already mentioned *R-CNN*[26], the *Fast R-CNN*[36], where the authors added a parallel branch for predicting segmentation masks on each object detected with the regions of interest. This means the segmentation task does not depend on the results of object detection and the final result is the merge of these two branches results.

2.1.4 Datasets and Benchmarking

Multiple datasets have been proposed with the objective to benchmark several neural networks architectures. Some examples are *Cityscapes*[37], *KITTI*[38] and more recently *KITTI360*[39] applied for road image segmentation and *COCO*[40] and *PASCAL VOC*[41] for common object segmentation (examples of each presented on Figure 23. Although these works only focus on the segmentation of images, there is also a big interest on the segmentation of point-clouds using LiDAR information, as mentioned previously.

Looking upon the road-scene datasets, these are made of images, organised in scenes of continuous driving, having for each image a mask where each pixel has as value the identification of the class the object belongs to. It is important to note that, although there are sometimes more than 30 classes to be classified, these are simplified in order to reduce the complexity of the problem and to focus on the most relevant objects on the image.

(a) *Cityscapes* sample[37].

(b) *KITTI* sample[38].

(c) *KITTI360* sample[39].

(d) *PASCEL VOC* sample[41].

(e) *COCO* sample[40].

Figure 23: Samples from multiple open-source datasets.

Apart from real data, there have been attempts to make use of artificial data to increase the training performance of the models. For example, a group once proposed an algorithm capable of capturing data from the game *GTA V*[20], where it is possible to drive a car on a close to real-life environment, such as LiDAR information and road images, obtaining hundreds of additional data that in the end increased the performance of their model.

2.2 Discussion and Analysis

In this chapter, multiple techniques were presented and briefly explained, focusing on some of the state of art algorithms and their unique features while presenting the solution chosen to be the starting point for this work. Having analysed different architectures, it became clear what were the important features responsible for the boost of performance, as well as provided ideas for us to propose further improvements.

Although Semantic Segmentation and Instance Segmentation are closely related, we decided to focus on the first as the latter is an extension of the other, requiring more resources in order to run it while also adding an extra layer of complexion. Object Detection could also have been chosen but given the ease of usage and the existing libraries for this task compared to Semantic Segmentation, it made less sense from a deep exploration POV.

	Cityscapes mIoU	**PASCAL_VOC mIoU**
Normal Convolutionals	65.3	62.2
Normal Convolutional with CRF	-	72.0
DeepLabV3	81.3	89
Hierarchical MSA	85.1	-

Table 2: Performance of the state of art algorithms researched, in two datasets[42].

The metric applied was mIoU, which corresponds to the average IoU, that is, the area of intersection between the predicted segmentation map and the ground truth, divided by the area of union between the predicted segmentation map and the ground truth, because it is one of the main metrics used for this type of classification as it cannot be biased by the imbalance of classes available in an image (more on that on Section 4.1.1) and also because it was the one available on [42] as others such as the *Dice coefficient* (similar to IoU, memory consumption, time of execution and number of parameters were not available for comparison of algorithms between datasets.

Focusing then on the main Semantic Segmentation algorithms analysed, the most promising one was *DeepLabV3*[1]. We compared the main results regarding the use of this architecture and others on two state-of-art datasets: *Cityscapes*[37] and *PASCAL_VOC*[41], presented on Table 2. As can be interpreted, the use of CRF gave an increase of 10% to the standard convolutional architecture, with the added ASPP on *DeepLabV3*[1] increasing 17% compared to using CRF. Unfortunately, the *Hierarchical MSA*[35] does not have much more data on benchmarks apart from *Cityscapes* (opposite of *DeepLabV3*[1]) so we can not draw many conclusions about their performance on different contexts, but having the latter inserted on many more different data contexts with similar performance versus *Hierarchical MSA*[35] on the one dataset we are currently focusing means we don't really lose any considerable performance and have more choices regarding how we can use this model in the future. As a bonus, *DeepLabV3*[1] was a target of high research until now which provides a good source for accessing previous works for comparison/starting points.

Chapter 3

Preliminary Studies and Approach

3.1 Preliminary Studies

In order to gain insight on the subject, multiple state of the art solutions were analysed, focusing on the detection and classification of objects through the use of visual cameras or LiDAR sensors. Based on this research it was possible to list and identify relevant open-sourced datasets to be applied in this research.

Deeplab[1] was the most promising solution to be implemented and applied due to its results being one the best so far, as well as having a lot of support from the scientific community. In addition, the algorithm provides mechanisms to customise its backbone and multiple parameters, which can be adapted for exploring its potential on different contexts.

Given the large amount of data available for road scenes segmentation, it was decided to choose one that could be explored using other techniques on a future work. In this case, *KITTI360*[39] is composed of data from multiple synchronised sensors, from RGB cameras (examples presented in Figure 24), LiDAR and even *GPS*. Although only the camera sensor outputs will be used, this work could be expanded and benchmark the impact of LiDAR for segmentation, providing access to performance and cost comparisons given the same context. One of the main drawback of using this dataset is not many researches refer to it about its performance on multiple state of art models, as it is the case with *DeepLab*. Although there is one other dataset with the similar data composition, *KITTI*[38], it is older and the support for it has been diminishing. Also, the quality of the proposed dataset is higher compared to others by having higher quality images, being more context aware and overall organisation with easier access to manipulate.

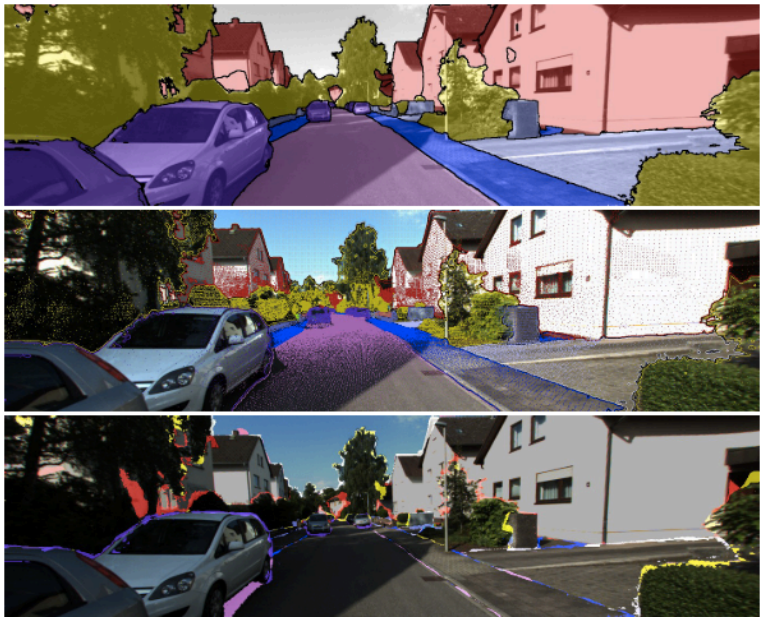


Figure 24: Sample of the *KITTI360* dataset[34]. (Top) Image with corresponding mask overlayed; (Middle) Image with corresponding LiDAR mask overlayed; (Bottom) Image with corresponding error mask from the labelling process.

3.2 Proposed Approach

This research proposes an attempt on applying an optimised version of this algorithm for image semantic segmentation on road scenes for future use on autonomous vehicles. For this, it will be applied the network *DeepLabV3*[1] with two different backbones, applied on the *KITTI360*[39] for training and testing, followed by some different optimisation techniques in order to reduce computation costs, while keeping the best accuracy possible.

3.3 Preliminary Results

As a first step it was decided to analyse the composition of our chosen dataset, *KITTI360* in order to collect information about its structure and types of data included. Using helper programs developed by the owners of the dataset, all data was downloaded and organised according to the documentation:

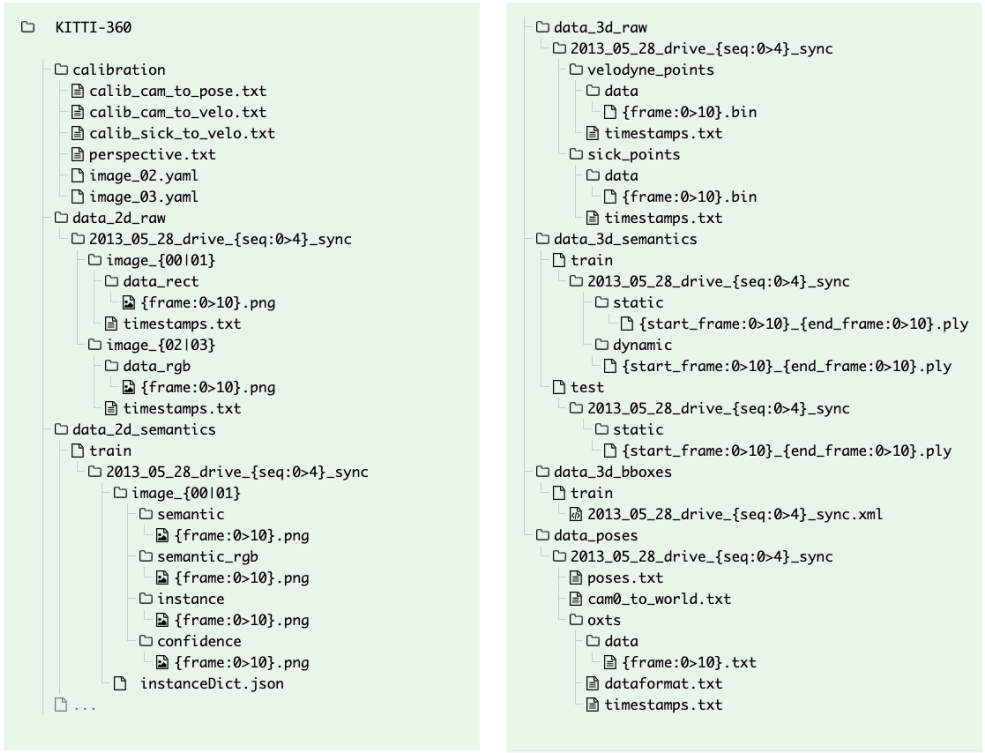


Figure 25: Folder structure for the data, according to the documentation.[1]

The data is composed of continuous sequences of images, organised in multiple scenes (different driving episodes), with each frame being named after a unique identification number. These images come from two different sets of sensors: a pair of *RGB* cameras on the front of the vehicle and one fish-eye camera on each side of the vehicle. In our case, it was decided to use one of the front camera sensors due to: (1) being a pair of cameras, the data is extremely similar on each side given the fact they are so close to each other, both visualising the front of the vehicle; (2) for the vehicle, its more important to keep track of what is in front of it compared to the side view; (3) given the type of camera on the side of the vehicle, a fish-eye, it would be more complex to pre-process these images. For these, only the data inside *data_2d_row* and *data_2d_semantics* will be used, as these correspond to the 2d images from the camera sensor and respective output masks, visible on Figure 25.

After understanding the dataset, it was needed to choose which scenes would be used for training, testing and validating our models. Usually, similar datasets already propose which scene is used for each step, but the group responsible for this dataset have yet to suggest a similar instruction. Our approach then was to analyse the composition of each scene in order to calculate the proportion for each entity on the image, as demonstrated on Figure 27, using the more balanced ones to train and the rest for training and validation. The reason so many unlabelled pixels exist is most of the original labels from the dataset are not used, to match with the *Cityscapes* format. These labels are a more detailed and specific version of the base features used, e.g. park, tunnel, vending machine and others.

[1] Source: KITTI360 website

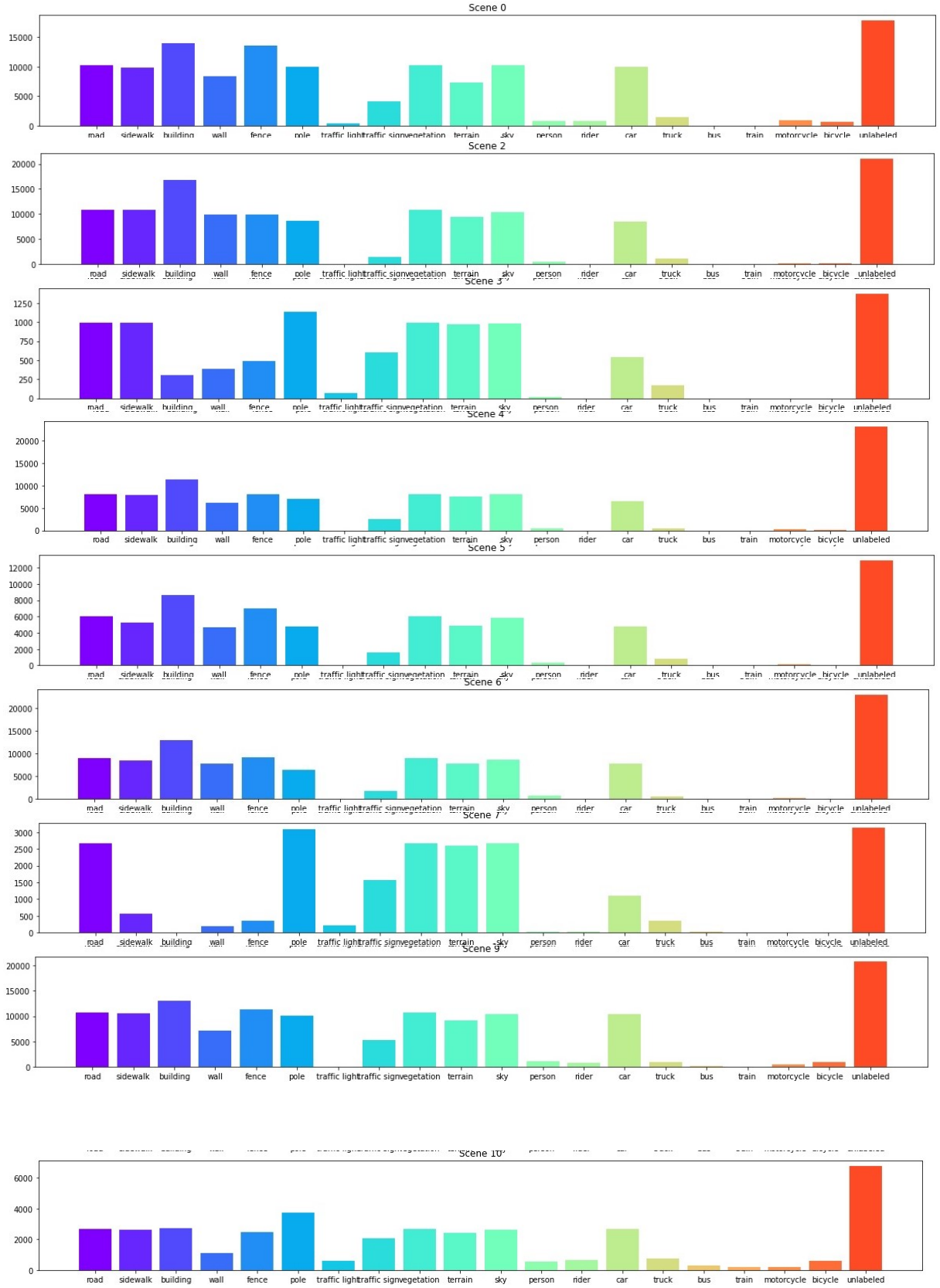


Figure 27: Number of each type of entities per scene.

As interpreted on the previous Figure, scenes 0, 4, 6, 9 and 10 seem the most suitable due to having the highest number of each entity represented, leaving 2 and 3 for testing and 5 and 7 for validation.

It was also developed a way to utilise a first iteration of the already trained model *DeepLabV3*[1] in order to test the proposed dataset, becoming a starting point of the work. The Figure 28 shows the output from an image from one of the scenes from the mentioned dataset.

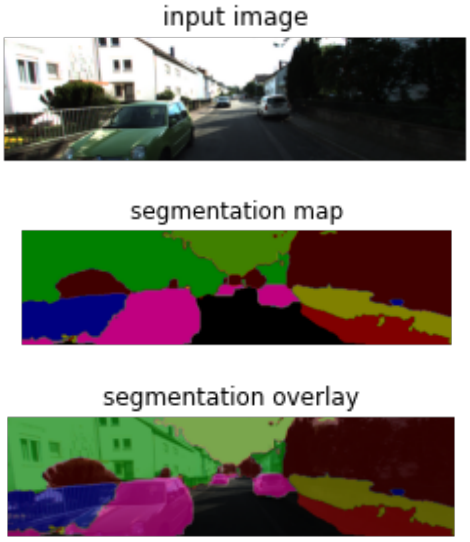


Figure 28: Output of a first iteration of the base *DeepLabV3*[1] model, with one image from *KITTI360*. (Top) input image; (Middle) output mask; (Bottom) input image with overlayed output mask.

The figure 28 represents the output of a small piece of code responsible for applying the algorithm on a target image. This was executed using the platform *Google Colab*, a cloud service capable of processing machine learning algorithms with access to a free GPU, although for the execution it was not necessary the use of it as it was a one time prediction. The machine is composed of 2 x Xeon 2 2.20GHz CPU, 12GB of Ram and a Nvidia K80 GPU, which ran the prediction is 2.5 seconds. We can see on the segmentation map that overall already predicts with high confidence, having more difficulty the more darker zones of the image. In the future we will attempt to mitigate this problem by increasing the brightness of the image.

Chapter 4

Results and Findings

4.1 Benchmarking and Improving the performance of DeepLab on Kitti360

Having the dataset and its splits defined (train, test and validation subsets), we benchmarked the latest weights trained by the original developer of *DeepLabV3*, resulted from training on the dataset *Cityscapes*, with a similar context to the chosen dataset.

4.1.1 Metrics

In the task of Semantic Segmentation, multiple evaluation metrics are used to access its segmentation performance and compare solutions, such as mIoU, also known as *Jaccard Index*, *DICE Score* (also know as *F1 Score*) and finally *Pixel Accuracy* [43]. The *Jaccard Index* will be used as the primary metric as it represents the mean performance of classification for each class, being basically the same as *DICE Score* (the latter applies more weight to the True Positives, being the meaning of the output the same on both, while the *Jaccard Index* will always have lower value comparing to *DICE Score*). In addition to the *Jaccard Index*, we will also use confusion matrix to evaluate each class's performance. As as secondary metric we will use *Pixel Accuracy*, which can be misleading when certain classes present heavily outnumber others, making the metric biased, as well as the human eye to analyse if the output masks appears more consistent with the ground truth.

It is important to note it was decided to define a small number of classes as more important then others, in order to focus our improvement evaluation on more real-world situations. These will serve to better grasp if the improvements are real and significant and were determined by their relevance directly related to autonomous driving. The following list presents the classes with the highest relevance for this study case:

- Road;
- Sidewalk;

- Traffic Light;
- Traffic Sign;
- Person;
- Rider;
- Car;
- Truck;
- Bus;
- Bicycle;
- Motorcycle.

4.1.2 Training Pipeline

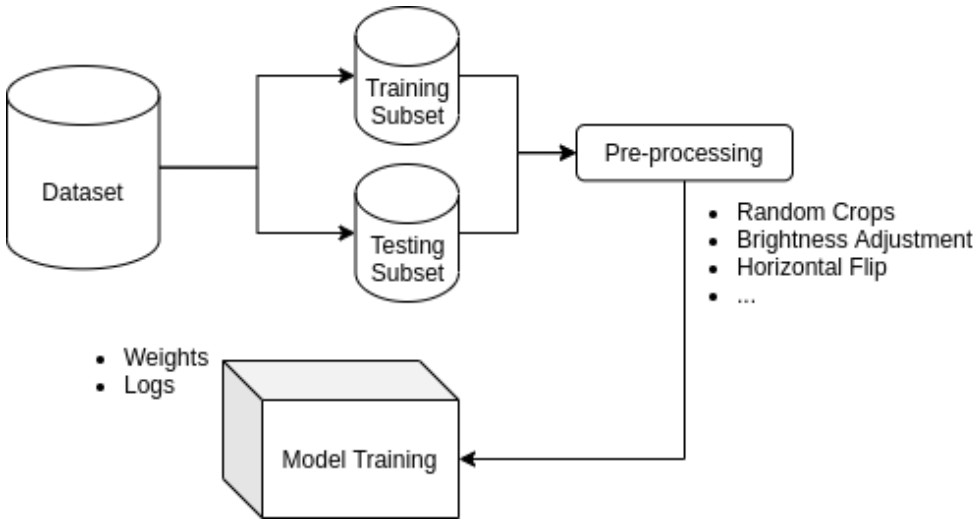


Figure 29: Training Pipeline.

As seen on Figure 29, the training pipeline consists of taking the training and testing subset of the dataset and apply a custom *ImageGenerator*, from the *Keras* library, capable of executing some pre-processing computations for each image, such as random crops, horizontal flips, and others. Currently, only random crops are used, because we have a high amount of data available, having even pixel normalisation not included on this pre-processing as the model already implements an initial layer for this computation. The training and evaluation of the models were held on a computer running a Ryzen 7 5800X, 32GB of RAM and a RTX 3080 10G resulting on a substantial speed up on training and prediction tasks compared to the previous machine from *Google Colab*. While training, the model is saved after each

epoch, while using *tensorboard* - framework to analyse training/validation metrics performance over time - and others support tools such as *EarlyStopping and ReduceOLRnPlateu* in order to prevent overfitting. Given the nature of the problem, it was used the loss function *Sparse Cross Entropy*, modified to ignore the unknown label (255) as indicated by the dataset developers. Finally, after some reports from other works[1], it seemed having the image cropped to a shape (375,513) instead of (512,512) resulted on similar performance with reduced computation time, so we opted for this approach as well.

4.1.3 Evaluation Pipeline

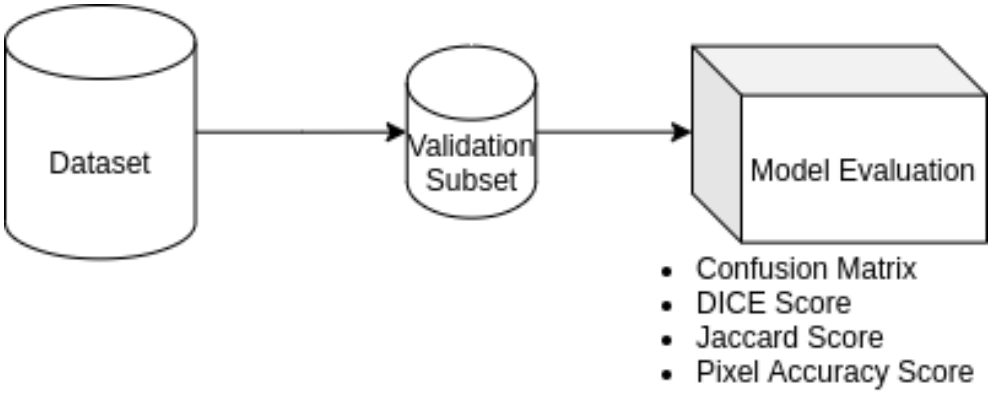


Figure 30: Evaluation Pipeline.

The evaluation consists on using the model to predict a subset of the dataset dedicated to validation, generating a final report with the different metrics used, detailed for each scene and each class, as well it's corresponding confusion matrix. This pipeline can be observed on Figure 30.

For human testing evaluation, a script was developed to predict random images from this subset and generate an image overlayed with the mask, easing the process of comparing results between multiple models.

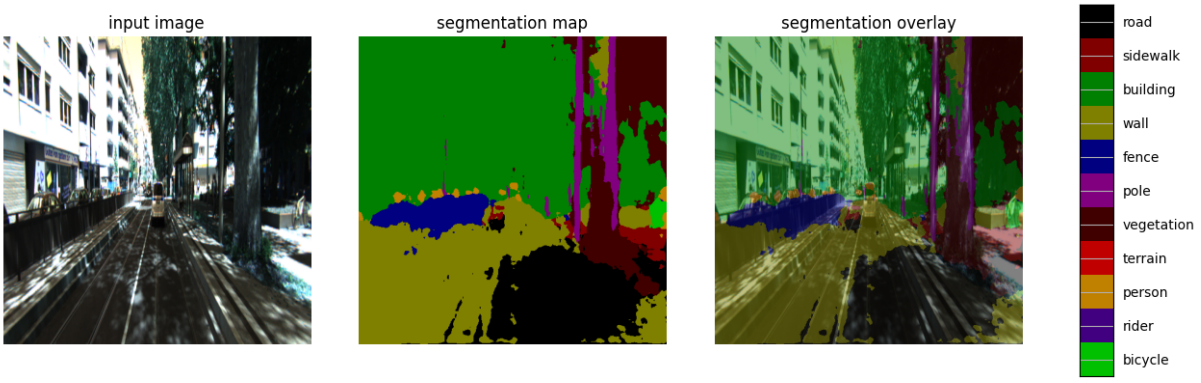


Figure 31: Example of a manual evaluation using the backbone *Xception*.

[1]Source: https://github.com/bonlime/keras-deeplab-v3-plus/issues/56

4.1.4 Models Results

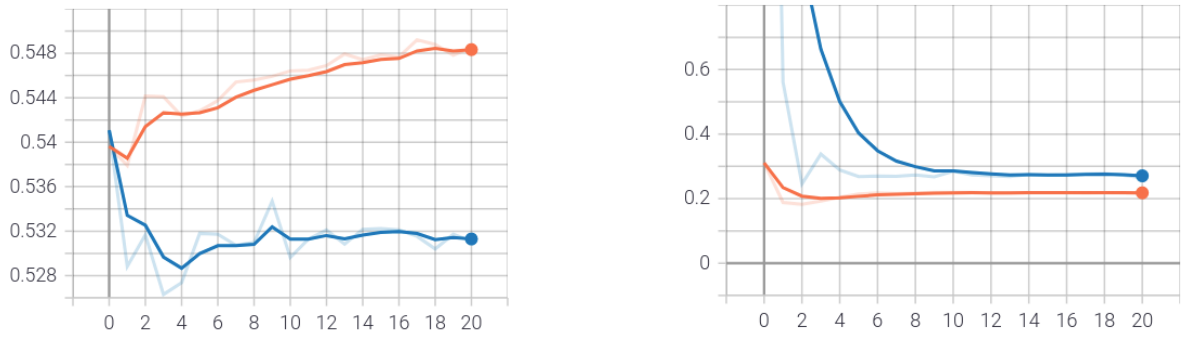
As explained before, the *DeepLabV3* exists with multiple variants, having different backbones for the model. At this time and for this work the backbones available were *Xception* and *MobilenetV2*, being the latter one the lighter version of the two.

On an initial stage, evaluations were executed for each backbone in order to retrieve information about their current performance on our dataset, such as mIoU, *DICE*, *Pixel Accuracy* and time elapsed of the evaluation. These would serve as a ground truth to determine if further training and optimizations would improve or not the models.

We started our evaluation on the analysis of the two available backbones. While their mIoU is extremely low compared to *Cityscapes* [1] due to poor image quality (high occurrence of shadows and different lighting conditions across the dataset, check Figure 32 for examples) and its different context as a more country area, as opposed to inner-city, it was naturally higher on *Xception* than *MobilenetV2*. As you can see on later images, both managed to be trainable and improved, being the *Xception* the model with higher performance, but only to a decent accuracy value. This proves that the dataset, on its current state, cannot be applied with full confidence, without some serious pre-processing of the images in order to minimise the noise for the network.

Figure 32: A cause for poor performance of the model, by different lighting and shadows[39].

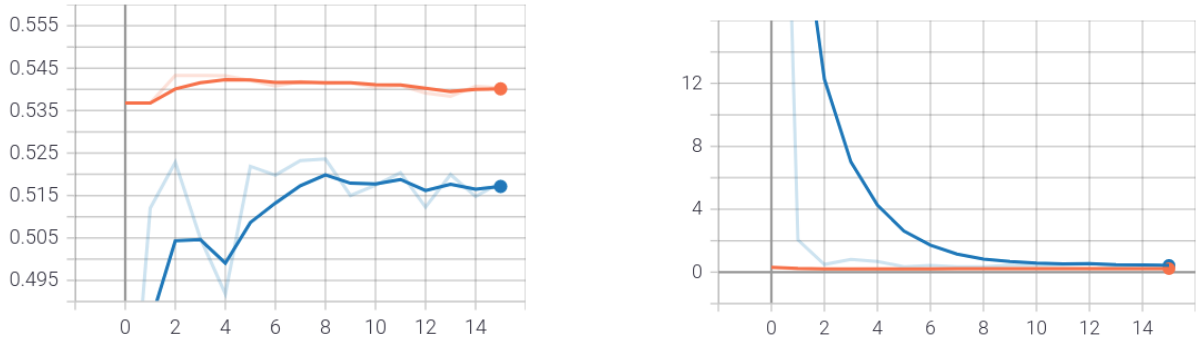
Initially, a learning rate of 0.0007 was being used which resulted on very poor results, misleading the network. After some trials, changing this value to 0.0001, made the network able to converge properly. The model training was executed for around 24 hours (taking 1 hour aprox. per epoch) and the maximum *Jaccard Index* values we could achieve on the validation set were 0,53% for the training of the full model (backbone and segmentation heads) and 0,51% when the backbone weights were frozen (as can be seen on Figures 33 and 34). Given the charts, we can safely assume the network learned correctly, reaching a state of plateau where no further improvements could be made. We can safely assume the network converged properly, as the learning rate was already very low.



(a) Jaccard index value by step of training.

(b) Loss value by step of training.

Figure 33: Training log for the *Xception* backbone, with unfrozen weights and learning rate at 0,0001. Orange line corresponds to the training and the blue line to the validation.



(a) Jaccard index value by step of training.

(b) Loss value by step of training.

Figure 34: Training log for the *Xception* backbone, with frozen weights and learning rate at 0,00001. Orange line corresponds to the training and the blue line to the validation.

The results were very positive for each class as can be seen on Table 3, presenting improvements on most classes. The improvements can be visually compared on Figure 37. Comparing the original and our trained model, we can see the latter adapted to correctly identify the roads and sidewalks despite the existence of shadows (Figure 35). On the other hand, the network has partially lost the ability to identify traffic lights, as shown on Figure 38, which we assume is the work of poor quality images and the resolution used for the images. Another important aspect is the choice to aggregate three classes - car, truck and bus - into one designated *vehicle*. The reasoning was the fact that the truck and the bus

appear very rarely on the dataset, which difficult the training and proper evaluation of these use cases. In general, the network always classified them as cars, being the class with the most similar features over all classes, which technically is not incorrect. More and better training data would be required in order to attempt separation of these three classes. The confusion matrix for the two models can be viewed on Figure 36.

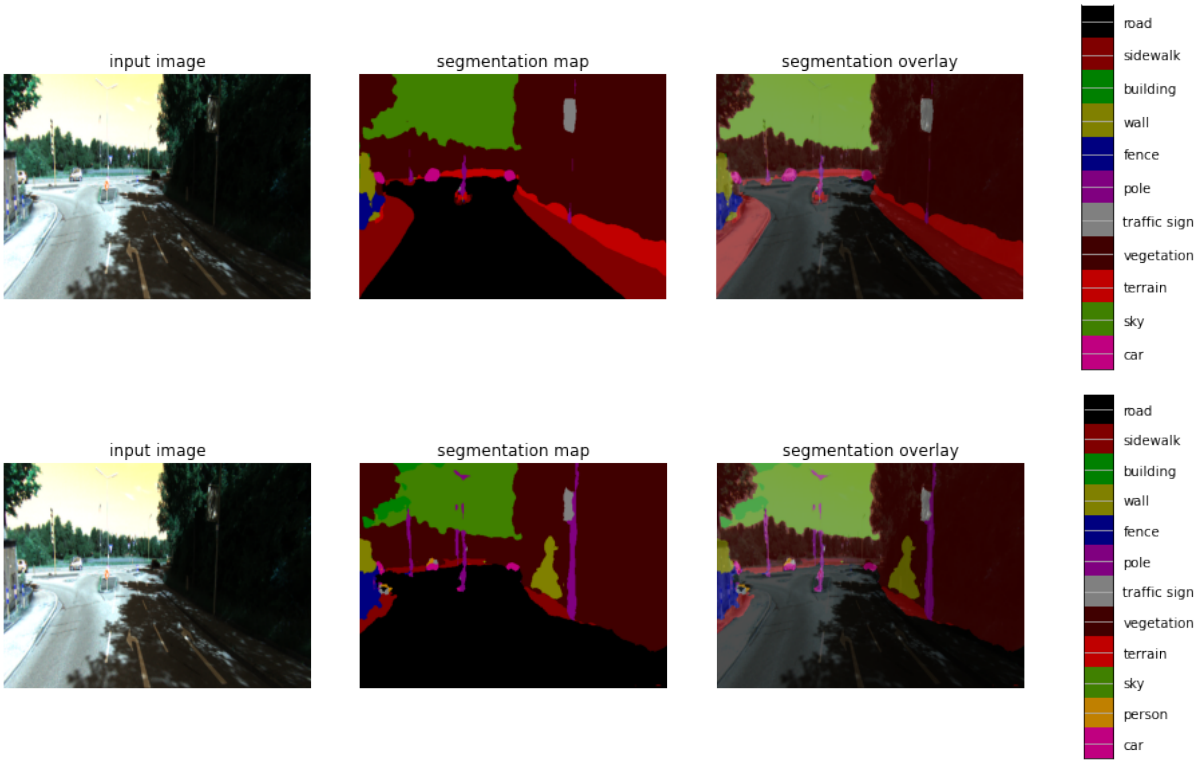
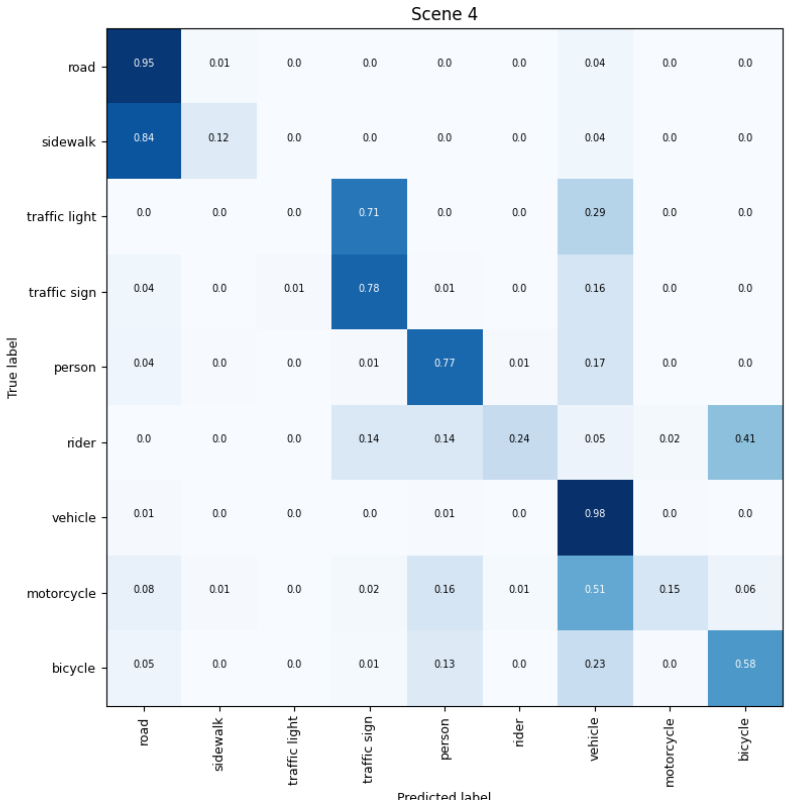


Figure 35: Comparison of the output from the retrained (Top) and base (Bottom) *Xception* model.

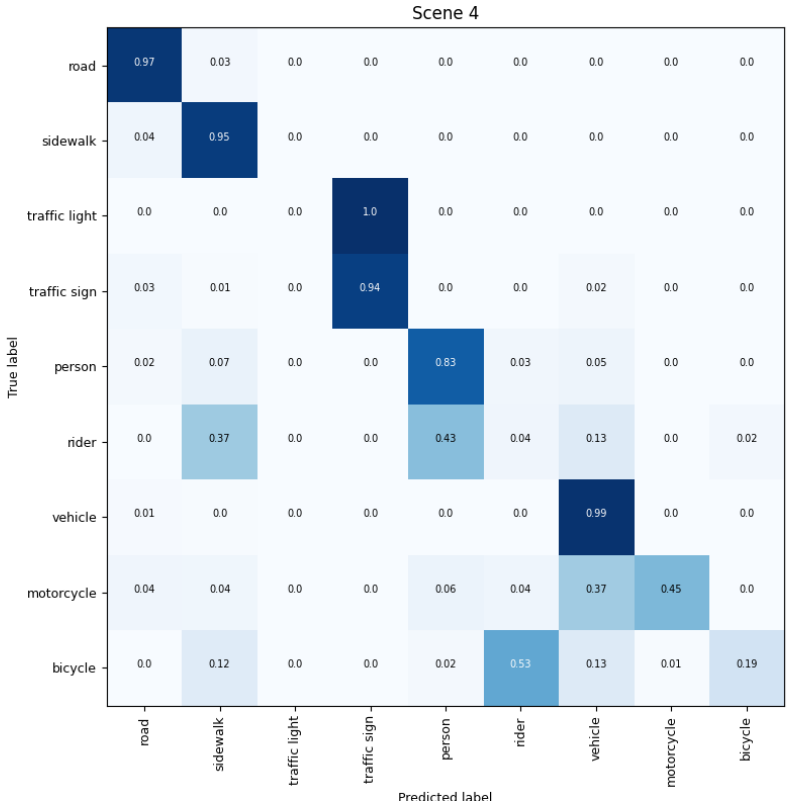
	Mean Pixel Accuracy	**mIoU (priority labels)**									**Mean mIoU**
		road	sidewalk	traffic light	traffic sign	person	rider	vehicle (car+truck+bus)	motorcycle	bicycle	
base xception	0.79	0.76	0.07	0.14	0.74	0.091	0.026	0.82	0.034	0.07	0.31
base xception with [0,1] norm	0.78	0.75	0.12	0.13	0.76	0.102	0.028	0.72	0.031	0.07	0.31
retrained xception	**0.96**	**0.95**	**0.87**	**0**	**0.89**	**0.749**	**0.002**	**0.95**	**0.22**	**0.08**	**0.645**
retrained xception with CRF	0.92	0.90	0.71	0	0.13	0.23	0	0.83	0.16	0.02	0.375
retrained and freezed xception	0.96	0.94	0.85	0	0.85	0.313	0.029	0.93	0.18	0.009	0.515
retrained and freezed xception with CRF	0.91	0.89	0.70	0	0.06	0.102	0.001	0.78	0.09	0.0002	0.295

Table 3: Results of the metrics before and after training, for the priority labels. As the test subset was composed of two different scenes, the value for each field is the mean of metric from each scene (except when it was not available on one of the scenes).

The use of CRF did not improve the results of network as expected, due to the correct fine detail the mask already possesses from the network.



(a) Confusion Matrix of the base *Xception*, on scene 4.



(b) Confusion Matrix of the retrained *Xception* also on scene 4.

Figure 36: Performance improvement visible using confusion matrix between the base and retrained model *Xception*.

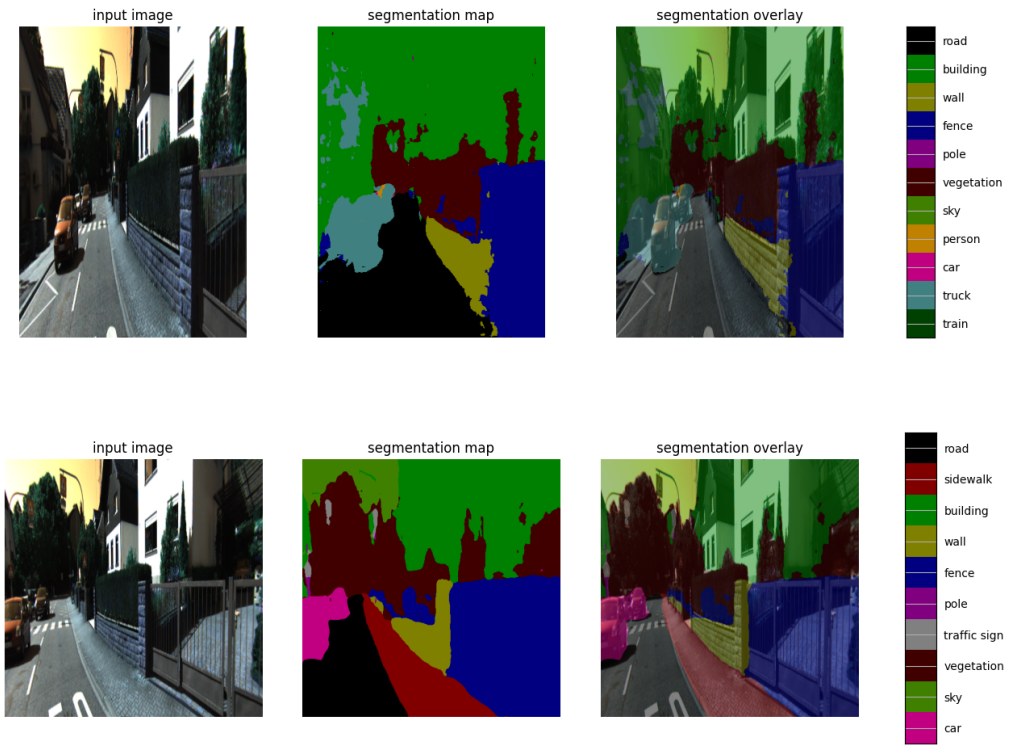


Figure 37: *Xception* (top) and retrained *Xception* (bottom) result on a random image.

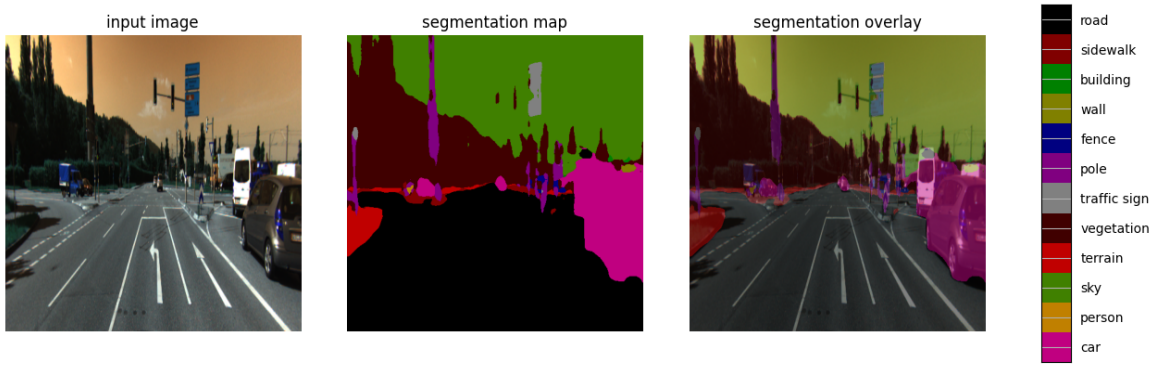


Figure 38: Result from the retrained *Xception* on an image with traffic lights.

As we can see on Figure 39, the front of the truck (top image) is classified as a car, given the proximity between the two, which is not good enough to safely say it is a good prediction. The same can be said about the bus (bottom image), which we can see it identifies the bus as a car, while later on the next frames, with a more close-up to the side of the bus, it starts classifying correctly. Treating both labels as a single one is the safer action in order to achieve a good level of consistency between predictions while not affecting the veracity of model.

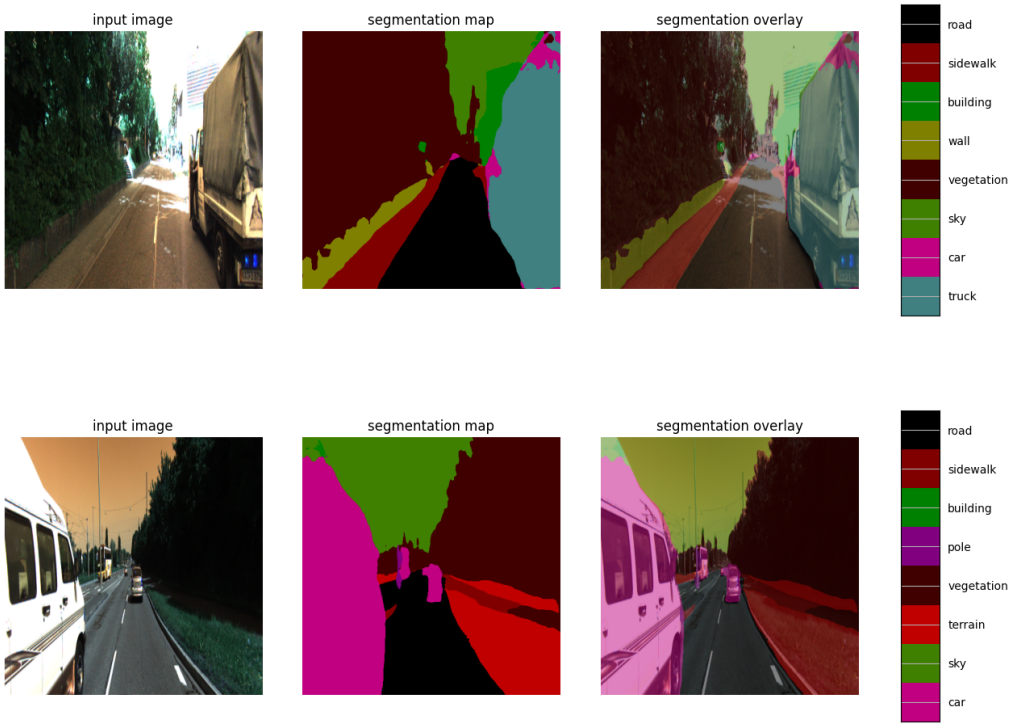
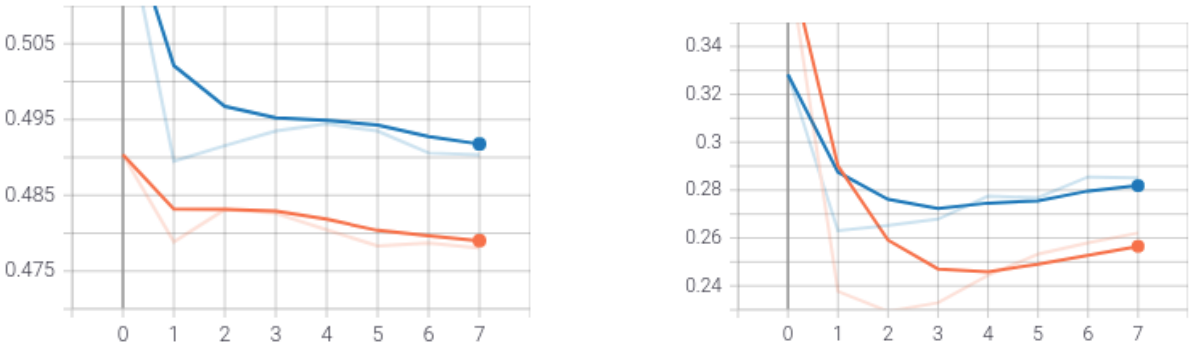


Figure 39: Result from the retrained *Xception* on an image with a truck (top) and a bus (bottom).

The fine-tune of the *Mobilenetv2*, opposed to the *Xception*, was smaller and quicker because the model would rapidly diverge around epoch 3/4, as seen on Figure 41, leaving little room for more improvements. Different experiments were made with different optimisation parameters, but the ones indicated on Table 4 were the best ones achievable. The reason the label traffic light disappeared from the graphic, differing Table 3, is because the *Mobilenetv2* lost the ability to predict it, so we opted to remove it to aid the analysis of the results.

	Mean Pixel Accuracy	**mIoU (priority labels)**								**Mean mIoU**
		road	sidewalk	traffic sign	person	rider	vehicle (car+truck+bus)	motorcycle	bicycle	
base mobilenetv2	0.60	0.61	0	0	0	0	0.24	0	0	0.09
base mobilenetv2 with [0,1] norm	0.60	0.61	0	0	0	0	0.24	0	0	0.09
retrained mobilenetv2	**0.95**	**0.93**	**0.81**	**0.79**	**0.2**	**0**	**0.94**	**0.02**	**0.12**	**0.48**
retrained and freezed mobilenetv2	0.95	0.92	0.81	0.79	0.17	0	0.94	0.02	0.15	0.48

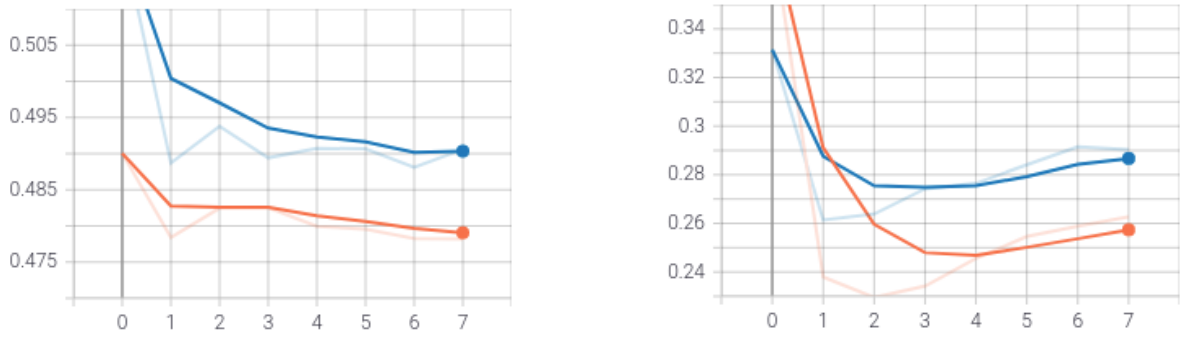
Table 4: Results of the metrics before and after training, for the priority labels. Same situation as Table 3.



(a) Jaccard index value by step of training.

(b) Loss value by step of training.

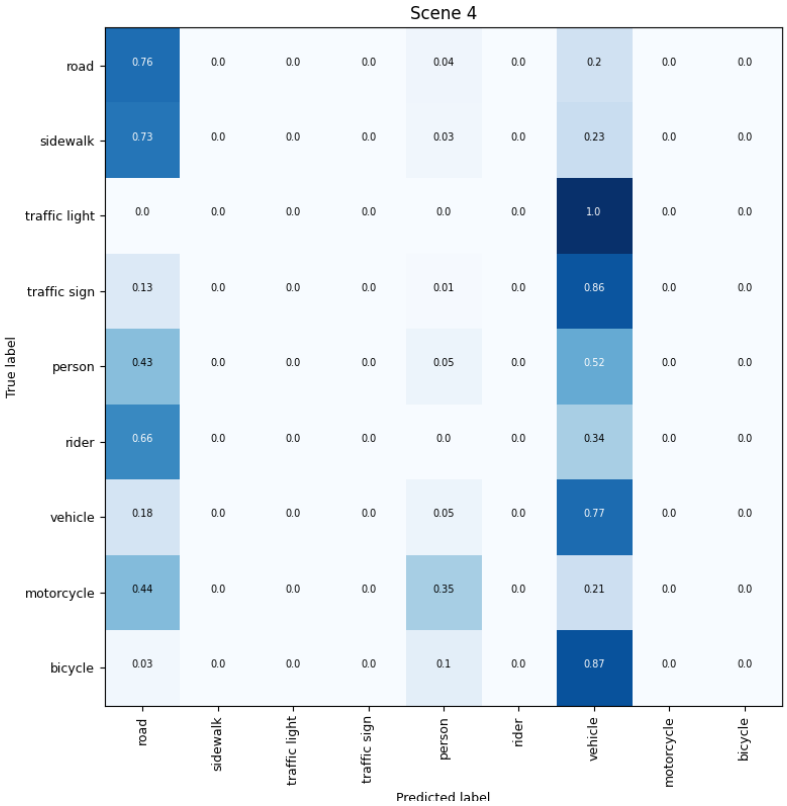
Figure 40: Training log for the *Mobilenetv2* backbone, with unfrozen weights and learning rate at 0,00001. Orange line corresponds to the training and the blue line to the validation.



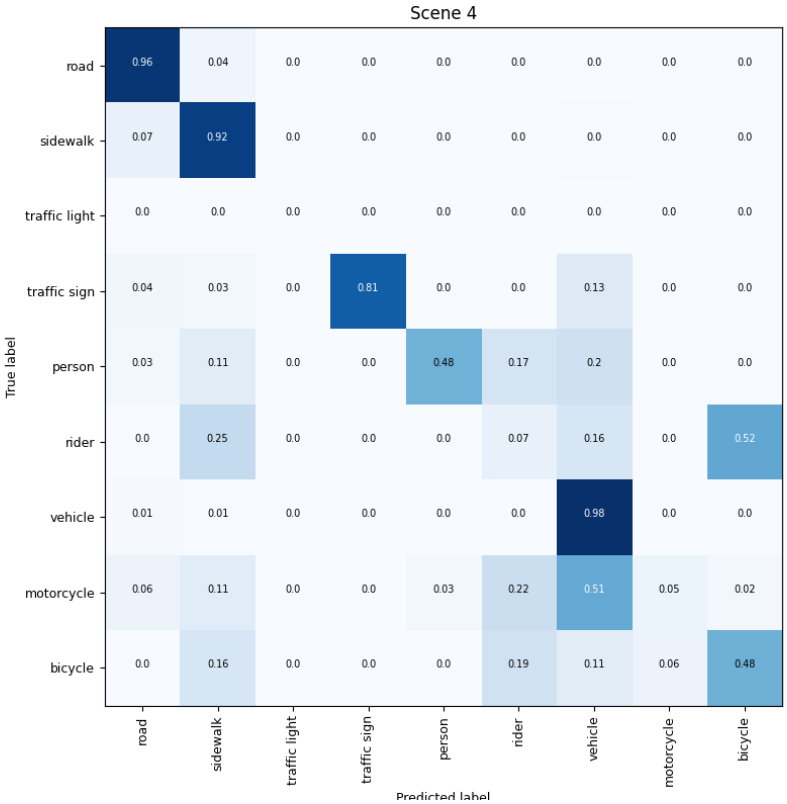
(a) Jaccard index value by step of training.

(b) Loss value by step of training.

Figure 41: Training log for the *Mobilenetv2* backbone, with frozen weights and learning rate at 0,00001. Orange line corresponds to the training and the blue line to the validation.



(a) Confusion Matrix of the base *Mobilenetv2*, on scene 4.



(b) Confusion Matrix of the retrained *Mobilenetv2* also on scene 4.

Figure 42: Performance improvement visible using confusion matrix between the base and retrained model *Mobilenetv2*.

As seen on Figure 42, initially the network did not perform good on the dataset, just loike *Xception*, but after training it became very capable of detecting the more important labels. Compared to *Xception*, due to its smaller size backbone, it has less ability to grasp some features from smaller labels, such as **persons and small vehicles (rider)** resulting on inferior results for these. But the major improvement comes from having a model much smaller produce results which comes close to *Xception*. An example of its improvement can be seen on Figure 43.

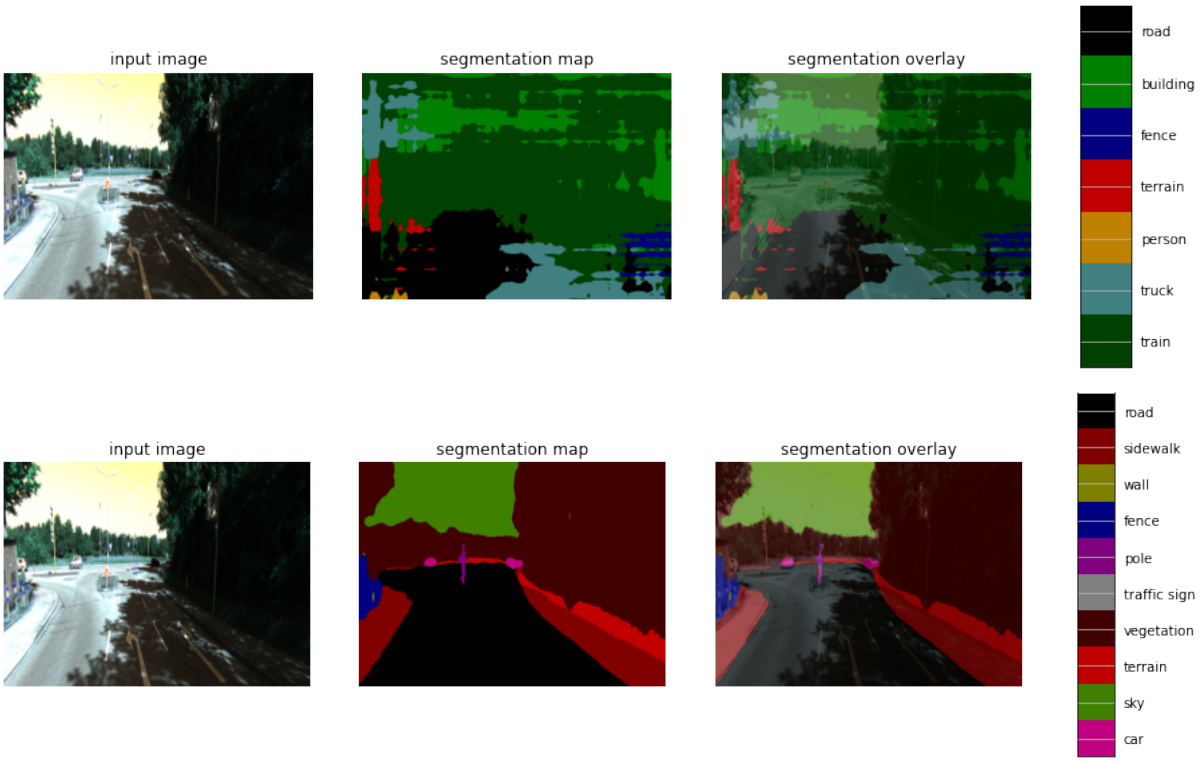


Figure 43: Comparison of the output from the base (Top) and retrained (Bottom) *Mobilenetv2* model.

4.2 Deep Neural Network Compression

While the network is being used on a high-end system, this would not be the case for its actual use case. Generally speaking, when applying models to more weak (but still powerful) systems such as mobile or embedded devices, these models need to be optimised to reduce their computation and resource costs. For this, different possible approaches exist such as:

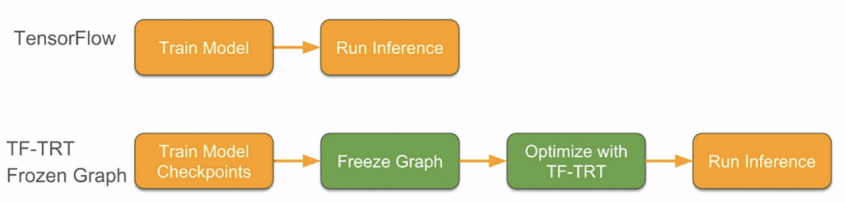
- Pruning: elimination of weights with low magnitude, not contributing significantly to the final output;
- Quantization: conversion of the network layers data bit representation, in order to reduce computation and resource costs;
- Knowledge Distillation: training a smaller network with the knowledge acquired by a bigger network in a compressed way;

For this application, we did not use knowledge distillation as it would require more experimentation and knowledge of the unique layers used on our model. As such, we attempted using all the other remaining techniques: *TensorRT*[2], a quantization library famous for its speed improvements, without compromising too much on the quality of the output; *TensorFlow Lite* - a lighter version of *TensorFlow* used in mobile or low powered devices, also used for quantization of neural network and a simple pruning approach. It is relevant to mention that this exploration was very superficial due to the complexity of the work and its high variety of possibilities. It would be unreasonable to try and perform extreme optimisations given the time frame available.

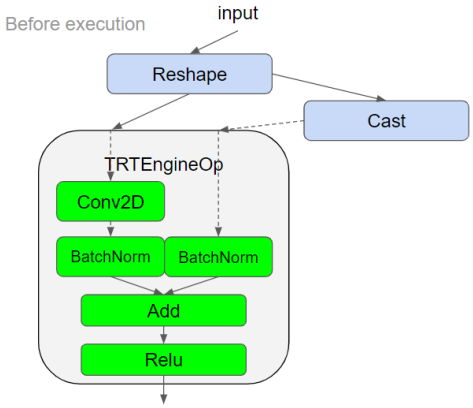
4.2.1 TensorRT

TensorRT consists of an framework developed by *NVIDIA* to optimise resources and inference time of deep neural networks. This method takes a neural network and converts each compatible layer to a high performance one (Figure 44), specific to the GPU used on inference, with lower bit precision, reducing its computational costs while also reducing the overall accuracy of the model, hopefully by just a small percentage. As seen in Figure 44b, the *TRTEngineOp* corresponds to an optimised batch of layers (green nodes), while keeping the unsupported layers (blue) as they are.

[2] *TensorRT* website: https://developer.nvidia.com/tensorrt



(a) TensorRT pipeline. We freeze the graph of the model and convert it using TF-TRT (tool for tensorflow model conversion).



(b) Conversion of a normal layer to a TensorRT optimised one.

Figure 44: TensorRT key aspects.[3]

In general, the inference time is reduced significantly (as seen in Table 5 and Table 7), but sometimes the model gets a high reduction on performance, as in our case. Although the parameters used only converted all *32bit* layers to *16bit*, having the possibility of going further, up to *int8*, the accuracy reduction was so noticeable that there was no point in trying to optimise it even further, having decided to stay on a **16bit** layer composition.

[3]Source: High performance inference with TensorRT Integration

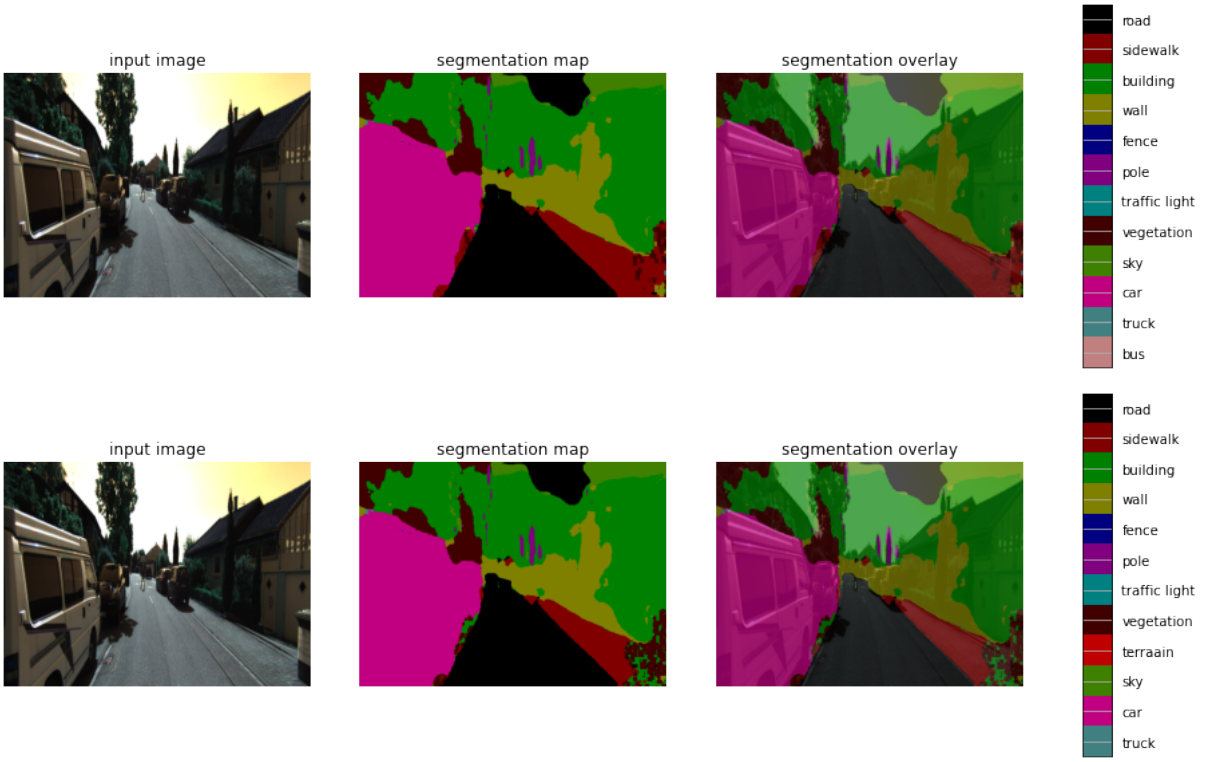


Figure 45: Comparison of the output from the base model (top) to the TensorRT model (bottom).

Visible on the label caption from both images in Figure 45, the *TensorRT* model has different labels and predictions than the baseline, and while they are minimal like random pixels - generally speaking they come close to each other. A weird behaviour encountered was that the type of input gave two different outputs between the two models. In more detail, our base model was trained using *OpenCV* as the library responsible for reading the images from the system and pre-processing them to feed the model and. When used while inferring the *TensorRT*, the outputs would differ slightly. If the library was changed for another such as *ImageIO*, the outputs would consistently be more similar between the models, but with overall accuracy reduction even higher than before. So we accounted this as a unique situation where, not only the *TensorRT* reduces the accuracy, but also the type of input associated. Other users reported similar situations [4]. In our case, to keep the consistency of the whole pipeline and model evaluation, we applied the results from *OpenCV*.

	Mean Pixel Accuracy	**mIOU (priority labels)**									**Mean mIOU**
		road	sidewalk	traffic light	traffic sign	person	rider	vehicle (car+truck+bus)	motorcycle	bicycle	
TensorRT Model	0.85	0.80	0.59	0	0.45	0.07	0	0.78	0.23	0	0.32

Table 5: *TensorRT* model metrics, with 16bit precision.

[4] Source: OpenCV vs ImageIO scenario

4.2.2 TensorFlow Lite

TensorFlow Lite is a watered-down version of *TensorFlow*, made to run on mobile or low-power devices. Given the application, it does not have GPU support and is optimised for CPU inference. For this reason, the fact the base model has multiple unique layers not originally supported by *TensorFlow Lite* and the sheer dimension of it, its performance is highly inferior to the baseline (Table 7), so it was decided as a non-viable option of optimisation.

4.2.3 Pruning

As explained earlier, pruning is a technique used in machine learning models, where the objective is to reduce computational costs without reducing too much of its accuracy, by removing redundant weights from the models. This is used in more popular machine learning algorithms such as *Decision Trees*, where some nodes can be removed, and, less popular, in neural networks, where we attempt to remove low-impact weights from the model (Figure 46).

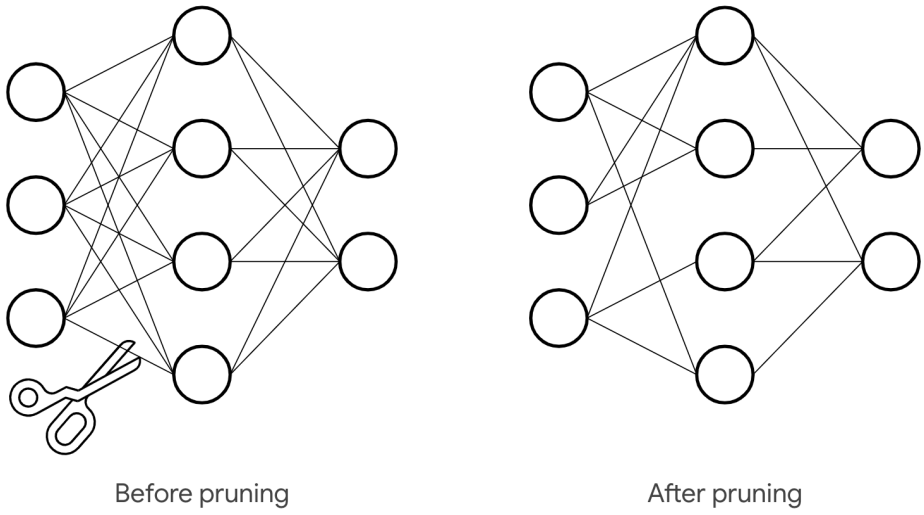


Figure 46: Example of the pruning technique.[5]

Given the incompatibility of some layers of our base model to be pruned, we attempted to prune only two layers, *DepthwiseConv2D* and *Conv2D* in 3 mismatch combinations. This method resulted in having, between the three different cases, the pruned layers with approximately 80% sparsity - of zero weights . The model size in the disk was one of the first improvements noticed, going from 450MB to 166MB.

[5]Source: TensorFlow Model Optimization Toolkit — Pruning API

	Mean Pixel Accuracy	**mIoU (priority labels)**								**Mean mIoU**
		road	sidewalk	traffic sign	person	rider	vehicle (car+truck+bus)	motorcycle	bicycle	
Retrained Xception	0.96	0.95	0.87	0.89	0.749	0.002	0.95	0.22	0.08	0.645
All layers Pruned	**0.92**	**0.90**	**0.68**	**0.64**	**0.2**	**0**	**0.93**	**0.009**	**0**	**0.376**
Conv2D Layers Pruned	0.93	0.91	0.75	0.64	0.15	0	0.92	0.01	0	0.38
Depthwise2D Layers Pruned	0.93	0.90	0.72	0.63	0.12	0	0.90	0	0	0.365

Table 6: Comparison between the three pruned models and the base retrained *Xception*.

Table 6 presents all the attempted combinations at pruning. Having only *Conv2D* layers pruned produced the best results overall, but we decided to keep the all layers pruned model as the best achieved because the *mIOU* was insignificantly lower while having slightly better prediction capabilities of detecting both vehicles and persons, despite the big reduction of accuracy from 0.645% to 0.376%. The labels that suffered the most appeared to be smaller ones in the images, such as *person*, *rider*, *motorcycle* and *bicycle*. Given the fact we are transforming low-impact weights on the network to 0, this behaviour was to be expected as the smaller features would be the most affected ones with the reduction of smaller details from the filters. Further experimentation on the sparsity value could be achieved, but the it would have been a very time-consuming task.

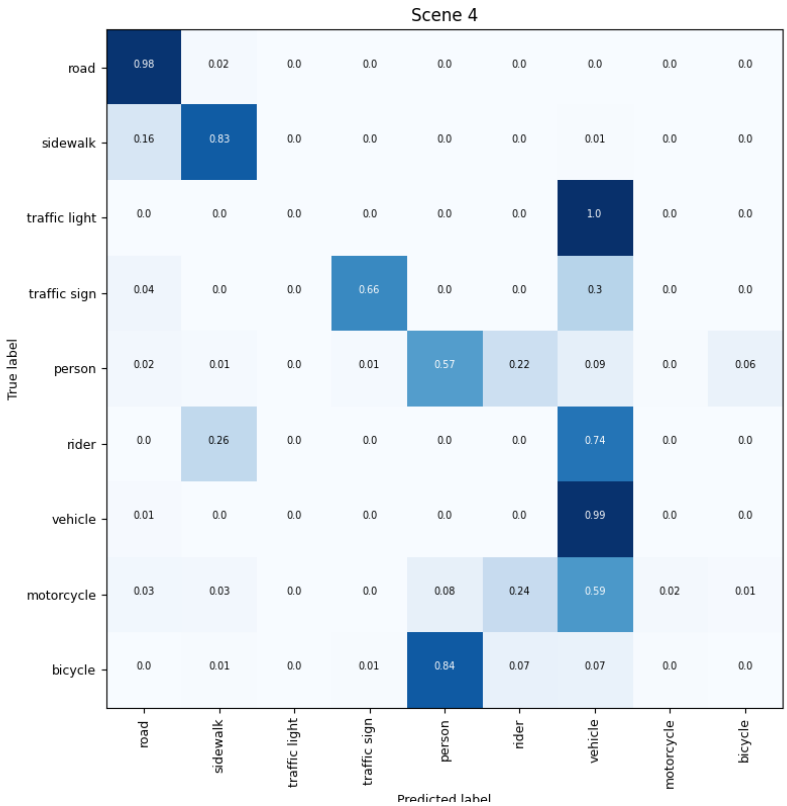


Figure 47: Confusion matrix of the *all pruned model* model. Compare with Figure 36b to verify which labels were more affected.

Analysing Figure 47 and comparing it to Figure 36b we can see that indeed, smaller labels such as *traffic light* and *person* were heavily affected with this pruning process. This can be verified on Figure 48.

Other classes like *motorcycle* and *bicycle* were almost completely removed from the models ability to be detected.

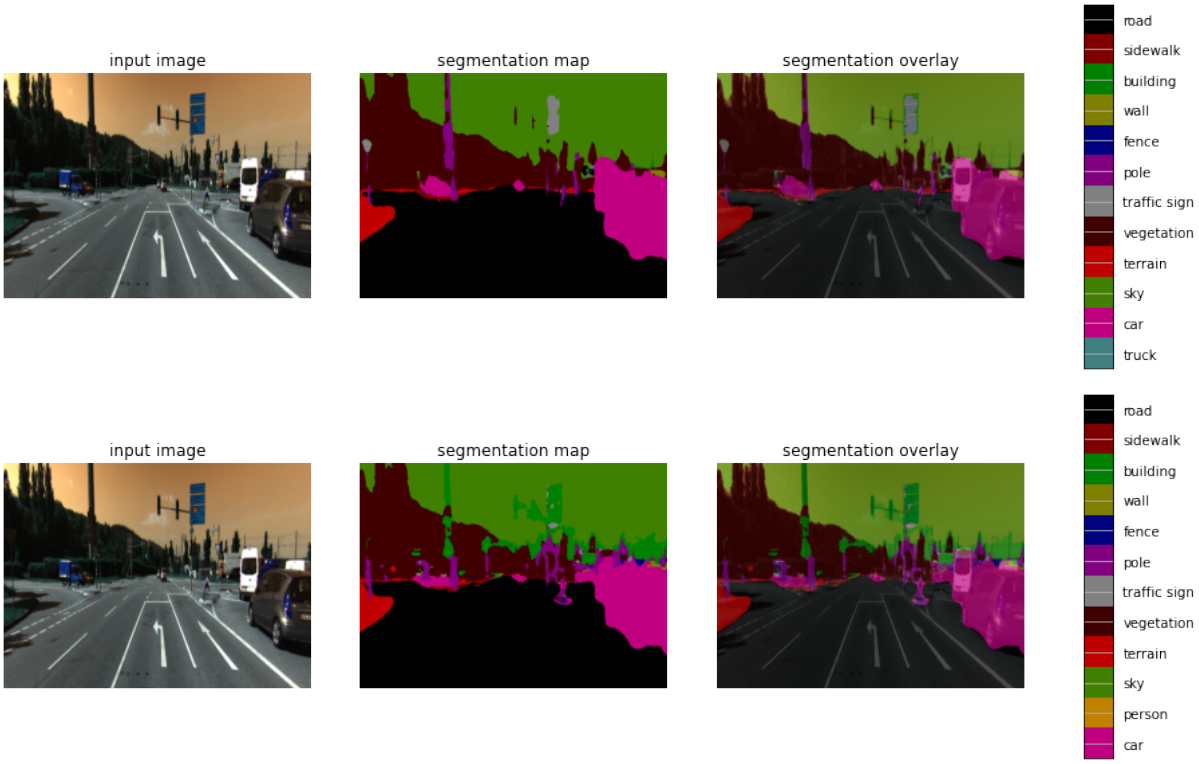


Figure 48: Comparison of the output from the retrained (Top) and pruned (Bottom) model.

4.2.4 Results Comparison

The comparison of the different models is presented in Table 7. The reason we did not compute the metrics for the *TensorFlow Lite* quantization was the high latency of computation associated with it. The results got slightly worst, but the high latency is the key factor, which makes it unusable for our use case. Also similar to Table 4, the label traffic light was removed to ease its interpretation.

The general results of the different optimisations were that the ability to segment the smaller labels, with smaller features, was the biggest problem. This does not appear as a big cost, given the reduced latency achieved, but given the smaller labels importance for autonomous driving, it is not in perfect condition to be applied in a real world scenario, without any kind of extra help from another algorithm.

	Retrained MobilenetV2	**Retrained Xception**	**Model loaded with TensorRT**	**Model loaded with Tensorflow Lite**	**Pruned Model (all layers)**
Time per inference	27ms	60ms	18ms	35.77s	**37ms**
mIoU (%)	0.48	0.53	0.33	-	**0.37**
mAcc (%)	0.95	0.96	0.85	-	**0.92**
GPU Memory Usage	172 MB	861 MB	566 MB	-	**885 MB**

Table 7: Comparison of the different compression techniques used in attempting to reduce the complexity of the model.

The GPU memory is increased on the *TensorRT* version, which is due to the duplication of some pipelines of the network, for fallback purposes while performing inference. To reduce this it would be required to further optimise each layer so it is fully converted to *TensorRT*. Also, the reason memory did not increase on the pruned model is these "pruned" weights still exist in memory, but are reduced to zero, which only accelerate the computation speed. For further inspection of the results from each model, please see Figure 49.


Input image
Base Xception
Retrained Xception
TensorRT Xception
Pruned Xception
Base Mobilenetv2
Retrained Mobilenetv2

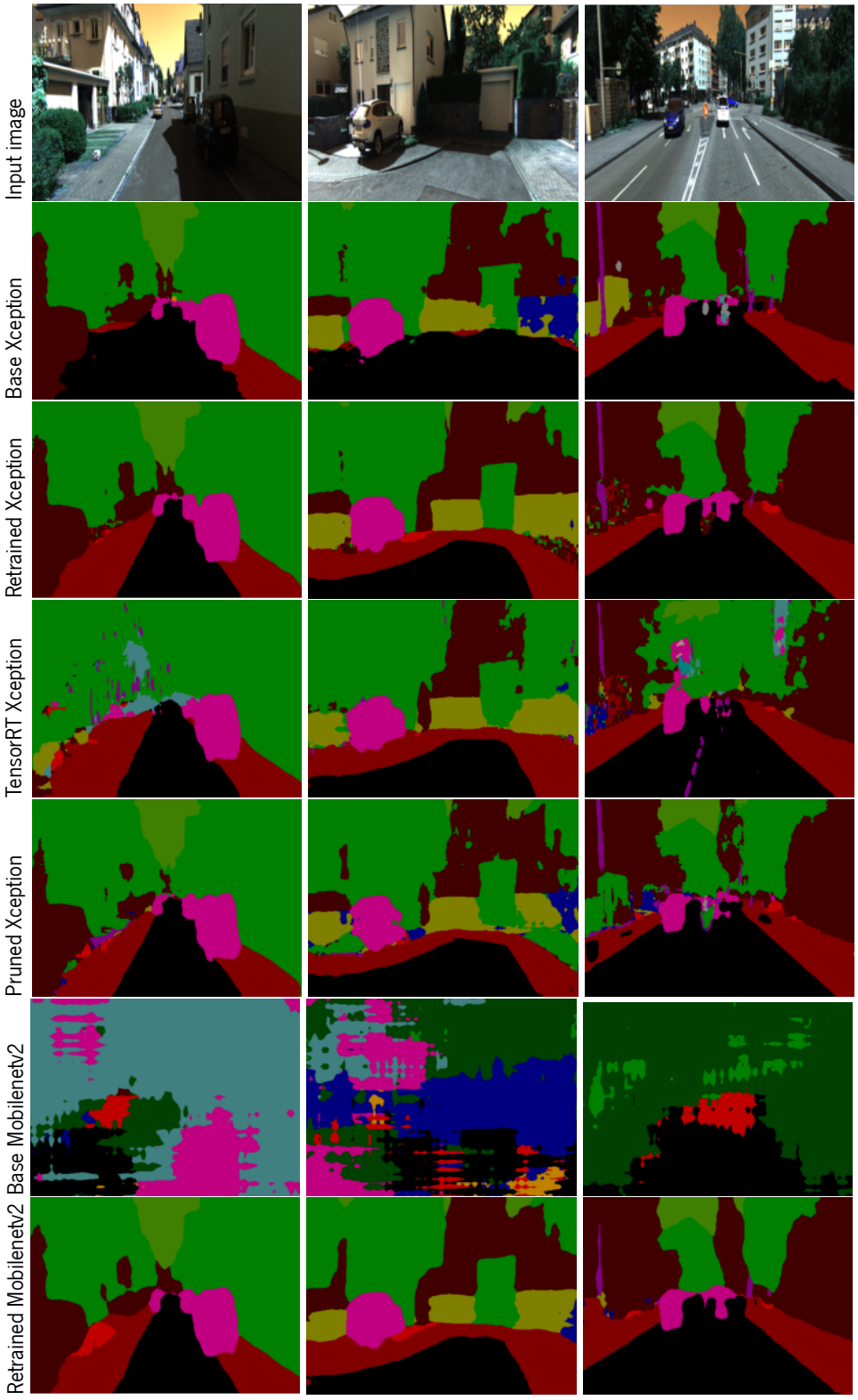


Figure 49: Comparison of results between the models.

Chapter 5

Conclusions

With a better understanding of the context of this work and its specifications, we came to multiple conclusions regarding what this field of research consists of, the main datasets and algorithms applied in the state of the art, and the definition of the next steps for this research.

First, the existence of semantic segmentation comes to provide a support on multiple areas, being autonomous vehicles the main focus of this research. Applied on road image segmentation, a few of the multiple open-sourced datasets for semantic segmentation exist, such as *Cityscapes*, *KITTI* and *KITTI360*. For the training and validation of our algorithms, the latter one is the one proposed to be used due to its recent date of publish as well as supporting *RGB* and LiDAR sensors.

Regarding the various available algorithms, only a handful were analysed in great detail due to the amount of existing state of art solutions. Specifically, the algorithms analysed were the *Unet*, *DeepLabV3* and *Hierarchical MSA*. *Unet* was responsible for a major breakthrough on image segmentation due to its downsample-upsample architecture, proving very efficient on biological image application but not so much on road image segmentation. A few years later, *DeepLab* appeared with unique features such as CRF and ASPP, which was constantly evolving, and *Hierarchical MSA* with its ability to parallel train and inference images with multiple resolutions to better locate and refine the boundaries of each entity on the image. Although the latter one reached better results on *Cityscapes* than the others, the improvement is negligible when the first provides more possibilities of exploration, with easier manipulation and execution.

As such, the *DeepLab* architecture was benchmarked and explored in order to achieve improvements on our segmentation task, ranging from accuracy enhancements and computation cost reductions, having as baseline our already described *KITTI360*. These optimisations were based on numerous techniques with the objective of, not only improving our model, but also to learn and explore state of the art solutions. Given the complexity of the matter, these were not deeply explored, but rather the understanding of which techniques work at a given situation.

Using the mentioned algorithm, we managed to achieve significant results on its performance, doubling the base mIoU. It is important to note the initial performance of the model was worst than the achieved by the original researchers, due to differences on the used datasets, having our *KITTI360* less quality than the standard *Cityscapes*, as well as a more country context, as opposed to inner-city. Both backbones used,

Xception and *Mobilenetv2*, proved to be quite capable on the task, being the first the one with better results due to the network size. However, *Mobilenetv2* achieved similar results with less fingerprint. Both could be used on a real context, depending on the degree of importance of some road entities, for example, riders are not detected by the *Mobilenetv2* and motorcycles are mostly undetected.

A final experiment was executed in order to attempt multiple optimisation techniques to minimize needed resources. Among these was a complex framework by the name of *TensorRT*, which converted network layers to a more optimised one, with less bits of computations. This was the most explored one, but due to its size and possibilities, we only scraped the surface of what it can truly achieve. Even so, we managed to collect interesting results with the optimised model taking 3 times less time than the original model while having a 33% of mIoU decrease. Also, using pruning we achieved a model with half the inference time and, again, 30% mIoU decrease.

The results collected showed the potential of Semantic Segmentation applied on a context of autonomous driving, but as explained before, it can also be applied to other situations. The state of the art showed us how powerful these are and how versatile and easy to adapt to other contexts of applicability with high success. Being an improved technique over the older, more basic (but still equally useful) Object Detection, this shows their importance in the context of Autonomous Driving for their ability to more accurately indicate where and what entities exist in one environment, being exactly why many already use it as first iterations of these self-driving vehicles. Although not explored deeply on this work, the usage of Instance Segmentation provides an extra layer of information, by distinguishing each type of unique entity. Of course, this comes at higher costs of implementation and, as explored, compression and optimisations are not easy to execute.

In the future, three different approaches could be optimised. (1) improve the model's performance and benchmark with the *Cityscapes* dataset - as a way to verify the model's perception performance on real-life situations; (2) invest on the neural network compression and optimization through the use of the *TensorRT* framework, by manually making each available layer's compatibility with the framework, if not already. The pruning method could also be improved on, as most of the layers used by our model are not compatible with *Tensorflow pruning* techniques. (3) on a more deep approach, use the LiDAR information on the dataset in order to train and validate a multimodal deep neural network based on the features extracted from *RGB* and LiDAR data. This possibility was one of the main reasons we opted for *KITTI360* as our target dataset.

Bibliography

[1] Liang Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(4):834–848, 2018. ISSN 01628828. doi: 10.1109/TPAMI.2017.2699184.

[2] Image Segmentation | Types Of Image Segmentation, Oct 2020. URL `https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python`. [Online; accessed 4. Feb. 2021].

[3] Mahnoor Ali, Syed Omer Gilani, Asim Waris, Kashan Zafar, and Mohsin Jamil. Brain tumour image segmentation using deep networks. *IEEE Access*, 8:153589–153598, 2020. doi: 10.1109/ACCESS.2020.3018160.

[4] Hongguang Li, Yang Shi, Baochang Zhang, and Yufeng Wang. Superpixel-based feature for aerial image scene recognition. *Sensors (Switzerland)*, 18(1), 2018. ISSN 14248220. doi: 10.3390/s18010156.

[5] Ciaran Robb, Andy Hardy, John H Doonan, and Jason Brook. Semi-automated field plot segmentation from UAS imagery for experimental agriculture. *Frontiers (Boulder).*, 11(December):1–13, 2020. doi: 10.3389/fpls.2020.591886.

[6] Da Hu, Hai Zhong, Shuai Li, Jindong Tan, and Qiang He. Segmenting areas of potential contamination for adaptive robotic disinfection in built environments. *Build. Environ.*, 2020. ISSN 03601323. doi: 10.1016/j.buildenv.2020.107226.

[7] Anil Chandra Naidu Matcha. A 2021 guide to Semantic Segmentation. *AI & Machine Learning Blog*, Jan 2021. URL `https://nanonets.com/blog/semantic-image-segmentation-2020`. [Online; accessed 29. Jan. 2021].

[8] Sorry, Elon: Fully Autonomous Tesla Vehicles Will Not Happen Anytime Soon, Jan 2021. [Online; accessed 29. Jan. 2021].

[9] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. 6 2015. URL http://arxiv.org/abs/1506.02640.

[10] Tsung Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42:318–327, 2020. ISSN 19393539. doi: 10.1109/TPAMI.2018.2858826.

[11] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014. ISSN 10636919. doi: 10.1109/CVPR.2014.81.

[12] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. A survey of deep learning-based object detection. *IEEE Access*, 7:128837–128868, 2019. ISSN 21693536. doi: 10.1109/ACCESS.2019.2939201.

[13] Abdul Mueed Hafiz and Ghulam Mohiuddin Bhat. A survey on instance segmentation: State of the art. 2020.

[14] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep Learning for 3D Point Clouds: A Survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, pages 1–1, 2020. ISSN 0162-8828. doi: 10.1109/tpami.2020.3005434.

[15] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A Survey of Autonomous Driving: Common Practices and Emerging Technologies. *IEEE Access*, 8:58443–58469, 2020. ISSN 21693536. doi: 10.1109/ACCESS.2020.2983149.

[16] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming Hsuan Yang, and Jan Kautz. SPLATNet: Sparse Lattice Networks for Point Cloud Processing. *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pages 2530–2539, 2018. ISSN 10636919. doi: 10.1109/CVPR.2018.00268.

[17] Andres Milioto, Ignacio Vizzo, Jens Behley, and Cyrill Stachniss. RangeNet ++: Fast and Accurate LiDAR Semantic Segmentation. In *IEEE Int. Conf. Intell. Robot. Syst.*, pages 4213–4220, 2019. ISBN 9781728140049. doi: 10.1109/IROS40897.2019.8967762.

[18] Larissa T. Triess, David Peter, Christoph B. Rist, and J. Marius Zöllner. Scan-based Semantic Segmentation of LiDAR Point Clouds: An Experimental Study. 2020. URL http://arxiv.org/abs/2004.11803.

[19] Alexandre Boulch. ConvPoint: Continuous convolutions for point cloud processing. *Comput. Graph.*, 88:24–34, 2020. ISSN 00978493. doi: 10.1016/j.cag.2020.02.005.

[20] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud. *Proc. - IEEE Int. Conf. Robot. Autom.*, pages 1887–1893, 2018. ISSN 10504729. doi: 10.1109/ICRA.2018.8462926.

[21] Can Chen, Luca Zanotti Fragonara, and Antonios Tsourdos. Roifusion: 3d object detection from lidar and vision. 9 2020. URL `http://arxiv.org/abs/2009.04554`.

[22] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. 2005. URL `http://lear.inrialpes.fr`.

[23] Tony Lindeberg. Scale invariant feature transform. *Scholarpedia*, 7:10491, 2012. ISSN 1941-6016. doi: 10.4249/scholarpedia.10491.

[24] Liang Zheng, Yi Yang, and Qi Tian. Sift meets cnn: A decade survey of instance retrieval. 8 2016. URL `http://arxiv.org/abs/1608.01807`.

[25] Mohamed Fouad, Hossam Zawbaa, Nashwa El-Bendary, and Aboul Ella Hassanien. Automatic nile tilapia fish classification approach using machine learning techniques. 12 2013. doi: 10.1109/HIS.2013.6920477.

[26] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. 3 2017. URL `http://arxiv.org/abs/1703.06870`.

[27] Pedro F Felzenszwalb and Daniel P Huttenlocher. Seg-ijcv. pages 1–26, 2015. ISSN 1573-1405. URL `papers3://publication/uuid/D1250C05-2FC7-4954-A734-E33EBBEECB95`.

[28] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. 12 2016. URL `http://arxiv.org/abs/1612.08242`.

[29] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. 4 2018. URL `http://arxiv.org/abs/1804.02767`.

[30] Siyuan Huang, Yixin Chen, Tao Yuan, Siyuan Qi, Yixin Zhu, and Song-Chun Zhu. Perspectivenet: 3d object detection from a single rgb image via perspective points. 12 2019. URL `http://arxiv.org/abs/1912.07744`.

[31] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, 9351:234–241, 2015. ISSN 16113349. doi: 10.1007/978-3-319-24574-4_28.

[32] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. UNet++: Redesigning Skip Connections to Exploit Multiscale Features in Image Segmentation. *IEEE Trans. Med. Imaging*, 39(6):1856–1867, 2020. ISSN 1558254X. doi: 10.1109/TMI.2019.2959609.

[33] Liang Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv*, 2017.

[34] Sik-Ho Tsang. Review: CRF-RNN — Conditional Random Fields as Recurrent Neural Networks (Semantic Segmentation). *Medium*, Mar 2019. [Online; accessed 29. Jan. 2021].

[35] Andrew Tao, Karan Sapra, and Bryan Catanzaro. Hierarchical Multi-Scale Attention for Semantic Segmentation. pages 1–11, 2020. URL `http://arxiv.org/abs/2005.10821`.

[36] Ross Girshick. Fast r-cnn. 4 2015. URL `http://arxiv.org/abs/1504.08083`.

[37] Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, and Stefan Roth. The Cityscapes Dataset for Semantic Urban Scene Understanding. 2016.

[38] A Geiger, P Lenz, C Stiller, and R Urtasun. Vision meets robotics: The KITTI dataset. The International Journal of Robotics Research. *Int. J. Rob. Res.*, (October):1–6, 2013.

[39] Jun Xie, Martin Kiefel, Ming Ting Sun, and Andreas Geiger. Semantic Instance Annotation of Street Scenes by 3D to 2D Label Transfer. *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2016-Decem:3688–3697, 2016. ISSN 10636919. doi: 10.1109/CVPR.2016.401.

[40] Tsung Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, 8693 LNCS(PART 5): 740–755, 2014. ISSN 16113349. doi: 10.1007/978-3-319-10602-1_48.

[41] Mark Everingham, Luc Van Gool, Christopher K.I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (VOC) challenge. *Int. J. Comput. Vis.*, 88(2):303–338, 2010. ISSN 09205691. doi: 10.1007/s11263-009-0275-4.

[42] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image Segmentation Using Deep Learning: A Survey. pages 1–22, 2020. URL `http://arxiv.org/abs/2001.05566`.

[43] Ekin Tiu. Metrics to Evaluate your Semantic Segmentation Model. *Medium*, Jan 2021. URL `https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2`.