Master Thesis on Intelligent Interactive Systems

Universitat Pompeu Fabra

Speed Traffic Sign Detection on the CARLA simulator using YOLO

Martí Sánchez Juanola

Supervisor: Federico M. Sukno, Dr.

July 2019

Master Thesis on Intelligent Interactive Systems

Universitat Pompeu Fabra

Speed Traffic Sign Detection on the CARLA simulator using YOLO

Martí Sánchez Juanola

Supervisor: Federico M. Sukno, Dr.

July 2019

Contents

Acknowledgement

Above all, I would like to express my sincere gratitude to my research supervisor Dr. Federico M. Sukno. Without his guidance, dedication, encouragement and extraordinary support this dissertation would have not been achievable. Also, I would like to express my special thanks to my office colleagues and PhD students, Adriana Fernández and Araceli Morales for assisting and advising me during the entire process of this thesis. In addition, I would also like to thank the Computer Vision Center of the Autonomous University of Barcelona for developing CARLA and to all the researchers and users that contribute to this platform making constantly improvements in the self-driving field.

I would also like to show gratitude to my family who have always been there for me with all the love and support that I needed during this project. Lastly, I am also extremely grateful to Sharon for proofreading my dissertation and for her support.

Abstract

Research on the self-driving and on the autonomous vehicles field is constantly increasing. The administrations are putting all their efforts to raise awareness among drivers about the responsibility that driving a vehicle implies. By their side, researchers are working hard to make vehicles as safe as possible in order to decrease the number of fatal accidents in our roads. Around 30% of these accidents have a key factor: speeding. Since some decades ago until our days, speeding has become a widely investigated topic by the large automotive companies. The European Union (EU) has recently announced new measures that will be taken in 2022 aiming to reduce car accidents. The most relevant one, is the measure that forces new vehicles manufactured at that year to have an intelligent speed assistance device integrated to make the driver aware when exceeding the speed limit of the road. Unfortunately, the oldest cars will remain without that system.

The aim of this project is to contribute to this research by integrating into a realistic driving simulator a system able to detect and recognise the speed traffic signs of the road, taking decisions that help the user make the driving process easier and safer. The simulator used in this study is the CAR Learning to Act (CARLA), an open-source simulator for autonomous research that mainly consists of two modules: the CARLA Simulator and the CARLA Python API module. To detect its road signs, a cutting-edge object-detection algorithm is used: the You Only Look Once (Yolo) algorithm. Instead of using a sliding window over several locations within an image, Yolo sees the entire image during the training and testing phases encoding contextual information about the object classes as well as their appearances. This characteristic allows it to be extremely fast while an image is being evaluated.

Identifying the speed traffic signs of a road can have a wide range of utilities. In this project, two applications are exposed: a warning application, to notify the user that the vehicle speed has overtaken the maximum allowed speed of that road, and a control application, to reduce the vehicle speed if it exceeds the road speed limit. Results show that the detection procedure is achieved satisfactorily with a precision

metric value of 0.92. Moreover, the system is tested both on the CPU and on the GPU, making it reproducible in most of the environments. Running it on the CPU takes a total time of 130 ms, while running it on the GPU, 8 ms are needed to evaluate the current CARLA scene and determine whether it exists or not any speed traffic sign.

Keywords: object-detection, self-driving simulator, traffic sign recognition, Yolo, CARLA

Chapter 1

Introduction

Over the last 15 years, fatality rates in the European Union roads have decreased more than 50%. Despite this progress, there are still 70 people a day who lose their lives on these roads [1] causing a life-changing impact on individuals and their families. All of the governments agree that some actions have to be taken in order to address the root cause of this problem aiming to give way, in the near future, to the ambitious era of safe and comfortable transportation [2] but there is certainly not a silver bullet. Violeta Bulc, the European Commission Transport Commissioner, states that 'a mix of measures is required: measures we know work well, like education and enforcement; as well as new and innovative solutions, especially when it comes to vehicles and infrastructure'. Referring to innovative solutions, different countries have recently made some massive investments in order to improve signage getting better road infrastructures [3], but in spite of these investments, most of the road accidents occur due to speeding. Speed is a key factor in around 30% of mortal accidents [4]. It is complex to predict the exact number of fatal accidents due to speeding, but it highly affects the risk of being involved in one of them: the higher the speed is, the less reaction time drivers have and it is more difficult to prevent an accident. Also, if an impact occurs, more energy is released and a part of this energy is absorbed by the driver and his or her companions. There are two ways of solving this: the first one is by raising awareness to the road users about the responsibilities

of being on the road and the second one is by making safer vehicles controlling their speed. In 2017, the Heracles Group of companies won the Excellence in Road Safety Award [5] launching the social responsibility program the 'Good Driver' project [6], designed for professional drivers. The initiative aimed to improve drivers' behaviour through a series of 'webisodes' which gave them useful tips related to a wide range of road safety topics faced by professional drivers in their daily routine. This program was well-accepted by the drivers of the company where it was tested: the results indicated that 85% of the drivers watched a video on a weekly basis. Although theoretical awareness sessions are highly recommended, humans are prone to fail and making mistakes on the road can be critical. For this reason, to build safer vehicles where the maximum speed could be controlled according to the allowed road-speed must be a priority for governments and researchers.

Over the last decades there has been extensive research on autonomous vehicles. The advances made from the first 'autonomous' car prototype (the American Wonder car) in 1925 until the most cutting-edge innovations recently developed by large companies such as Google or the Volkswagen Group Research have been huge and the results obtained by these advances have been widely accepted by the research community. But unfortunately the 'fantasticar' will take years to get there. Taking a look at the above mentioned cause of most of the accidents, it is reasonable to think that detecting the maximum allowed speed of a road and inferring to the speed module of the vehicle could help to avoid them. In fact, controlling the speed of a motor vehicle is something which has already been explored. The Cruise Control (CC) system was introduced in 1968 and has been widely used in many vehicles to determine their speed according to the speed set by the driver. Moreover, a more refined version of the CC system is capable to automatically adjust the vehicle speed to maintain a safe distance from vehicles ahead [7]. Regarding to the road signs detection and recognition, there is extensive research showing evidence of the strength of object detection techniques but the majority of studies follow the same pattern [8], [9]: extract a set of robust features from the input image (such as SIFT [10], Haar, HOG [11], convolutional layers...), take classifiers or localizers to identify the desired objects in the feature space and run them at various locations and scales

over the test image. Iterating a classifier thousands of times over an image demands many computational resources, prevents generalization and can introduce a micro-delay, turning the whole system into a pseudo-real-time system.

Based on this previous research, the main motivation of this project is to build a system able to detect and to recognise speed traffic signs in the wild, install it into a non-autonomous vehicle and control the maximum speed it can achieve according to the recognised speed traffic sign. To take control over the vehicle speed limit, the engine control unit (ECU) would be inferred as the CC system does but due to legal restrictions (any device hitched to a vehicle obliges the vehicle to pass the inspection exam), lack of awareness about how does the ECU work and the mere fact that developing and testing algorithms for autonomous vehicles in real world is an expensive and time consuming process, a simulator will be used. What also motivates this project is that all of the developed software tools that will be implemented into the simulator will be reproducible, free-available and open-source, contributing in this way to the autonomous driving research community. In order to detect and recognise the simulator speed traffic signs the system will take advantage of computer vision resources to compute object detection within an image. The chosen model must work in real-time, be open-source, have high accuracy results, be easy to use and easy to be integrated into a Python code and have a high capability to generalise. This last model's feature is an important requirement for this project because non-natural images from the simulator are going to be taken into account for the training, validation and test phase and they comprise a wide range of weather condition scenes. The state-of-the-art object-detection model that best fits all these mentioned requirements is You Only Look Once (Yolo) system [12]. Yolo sees the entire image during the training and testing phases encoding contextual information about the object classes as well as their appearances instead of using a sliding window over several locations in an image, so it uses a single neural network to predict bounding boxes and class probabilities directly from full images in one evaluation. In this way, running a classifier thousands of times over an image is avoided.

Figure 1: A CARLA Town street [13]

Concerning the simulator, there are large companies such as Microsoft or Toyota who are doing huge advancements in the autonomous driving research. Both have implemented their own open-source simulators built on the game engine 'Unreal Engine[1]' to support development, training and validation of autonomous driving systems. Microsoft launched in 2017 the high-fidelity visual and physical simulation for autonomous vehicles AirSim [14] and the Toyota Research Institute sponsored the Computer Vision Centre's simulator CARLA [13]. Both are very similar: they are both built on Unreal Engine, have flexible APIs where users can control all aspects related to the simulation and users can set different sensor suites including LIDARs among others. The reasons why CARLA simulator is going to be used in this project are that it has a large community of users/researchers to interact with, it has a planned roadmap[2] until December 2019, it is well-known by all the autonomous driving research community and it is developed by researchers from a Spanish university: the Autonomous University of Barcelona. As shown in Figure 1 CARLA provides realistic urban layouts (including buildings, vehicles, pedestri-

[1] `https://www.unrealengine.com`
[2] `https://github.com/carla-simulator/carla`

ans...) and a wide range of environmental conditions, which highlights the similarity with the real world. This high degree of similarity allows to replicate the system this project will implement to a real car.

1.1 State of the art

During the last few years and justifying the evidence that object detection is a core problem in computer vision, many models to recognise different road-signs have been developed providing high accuracy values. However, the majority of them employ the same techniques: extract a set of robust features from the input image (such as SIFT [10], Haar, HOG [11], convolutional layers...), take classifiers or localizers to identify the desired objects in the feature space and run them at various locations and scales in a test image. There are more recent approaches like R-CNN [8] that use region of interest (ROIs) proposal methods to first create potential bounding boxes in an image and then run a classifier on these proposed boxes but for both cases, running a classifier thousands of times over an image implies thousands of neural network evaluations to produce detections. This demands many computational resources, prevents generalization and can introduce a micro-processing-delay, turning the whole system into a pseudo-real-time system. What's more, it needs a post-processing phase to refine the multiple bounding boxes, eliminate duplicate detections and rescore the boxes based on the number of objects of the scene [15].

In 2016 Joseph Redmon et Al. presented You Only Look Once (Yolo) system [12], which is a new approach to object detection. Yolo is an unified structure that frames detection as a regression problem getting a not too complex pipeline allowing the system to be extremely fast: it runs images in real-time at 45 frames per second and processes streaming video also in real-time with less than 25 milliseconds of latency. Compared to other state-of-the-art detection systems, it makes more localization errors but it is less likely to predict false positives on background. It also learns very general representations of objects, outperforming other detection methods when generalizing from natural images to other domains like artwork or unexpected inputs.

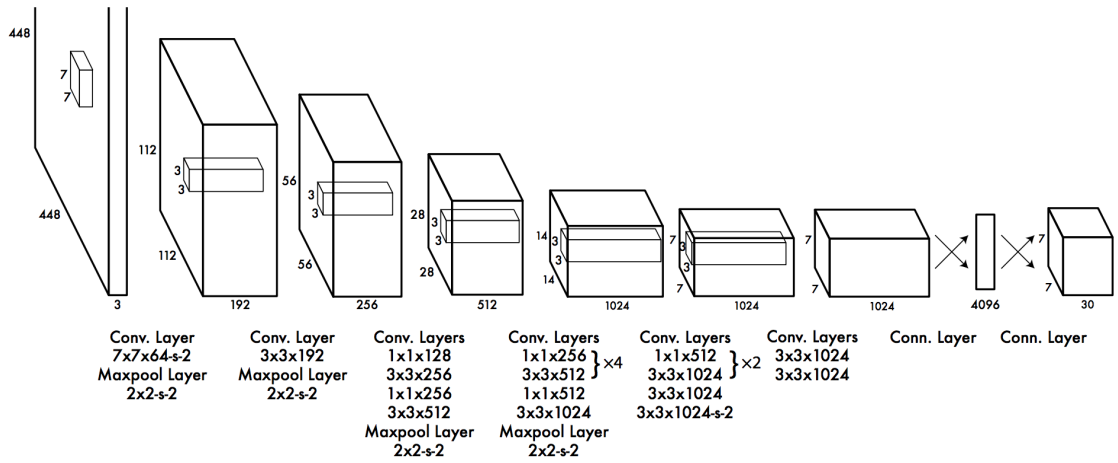

Figure 2: Yolo's Architecture [12]

Instead of using a sliding window over several locations, as some top detection methods like R-CNN do, Yolo sees the entire image during the training and testing phases, implicitly encoding contextual information about their classes as well as their appearances, avoiding doing some object detection mistakes in the background of the images. To perform the detection procedure, the system divides the input image into an S x S grid and a single convolutional neural network (CNN) predicts simultaneously B multiple bounding boxes for each grid cell, their confidence value and their classes probabilities for each region. The B bounding boxes are weighted by the predicted probabilities and the non-maximal suppression technique is applied to fix multiple detections when objects near the border of multiple cells are localized by these near-cells. Delving into the Yolo network design, its architecture is inspired by the GoogLeNet model for image classification [16] but instead of having 22 layer deep CNN, it has 24 followed by 2 fully connected layers. To reduce the features space from preceding layers, 1 x 1 convolutional layers are alternated between layers as shown in Figure 2. The convolutional layers are pretrained on the ImageNet [17] classification task at half the resolution (224 x 224 input image) and then it is doubled for detection. While detecting small objects that appear in groups, Yolo still lags behind state-of-the-art detection systems but both training and testing image sizes that are considered in this project are big enough to avoid these kind of struggles, so it is not considered as a huge limitation to take into account.

Several challenging benchmarking datasets of natural images such as the BelgiumTS Dataset, Cityscapes [18], GTSDB [19] among others and some synthetic images that will be obtained from the simulator will be taken into account to train Yolo.

Regarding the simulator, important technological companies such as Microsoft or the Toyota Research Institute have developed their own state-of-the-art and open-source autonomous driving simulator systems. Far away is the racing car simulator TORCS [20] released at 1997 that inspired researchers to make several improvements into the virtual autonomous driving field knowing that developing and testing algorithms for autonomous vehicles in real world is an expensive and time consuming process. On the one hand, Microsoft launched in 2017 the high-fidelity visual and physical simulation for autonomous vehicles AirSim. This simulator is built on the game engine 'Unreal Engine' being a platform for researchers to test with deep learning, computer vision and reinforcement learning algorithms for autonomous vehicles. On the other hand, the Toyota Research Institute sponsored the Computer Vision Centre's simulator CARLA, which is an open-source simulator for autonomous driving research also built on Unreal Engine. Both are very similar: they both have flexible APIs where users can control all aspects related to the simulation, they both provide realistic urban layouts, a wide range of weather conditions and also users can set different sensor suites and add them to the player vehicle getting some inputs from the environment.

The main difference between these two state-of-the-art simulators is that CARLA deals with a large community of users and researchers to interact with. The communication tools are: a Git Hub site[3], a Twitter account[4], a Discrod channel[5], and a Youtube channel[6] among others. Also, CARLA has a planned timeline until December 2019 and being developed into an important university allows it being well-known by all the autonomous driving research community. CARLA provides very realistic urban layouts (including buildings, vehicles, pedestrians...) and a wide range of

[3] https://github.com/carla-simulator/carla
[4] https://twitter.com/carlasimulator
[5] https://discordapp.com/invite/8kqACuC
[6] https://www.youtube.com/channel/UC1llP9ekCwt8nEJzMJBQekg

environmental conditions, which highlights the similarity with the real world.

1.2 Objectives

The motivation that leads to this project is to build and install a system using the Yolo model capable of detecting and recognising speed traffic signs in the CARLA simulator. This system must be reproducible both on the CPU and on the GPU and capable to run on real-time. Moreover, two applications will be deployed. First, a 'warning application' that will advice the user to reduce the speed of the vehicle if it is exceeding the speed limit of the road. Secondly, a 'control application' that is going to progressively reduce the vehicle speed if it is exceeding the maximum allowed speed of the road plus its 10%.

In addition, a large data set of CARLA images will be created in order to contribute to the large community of researchers and users that are constantly developing this driving simulator. Furthermore, all of the software that will be developed will be reproducible, free-available and documented.

Chapter 2

Methodology

In this chapter of the thesis, the procedure on how to design and train the object detection model and how to integrate it to CARLA is going to be explained. Apart from that, some deployed applications will be exposed as well.

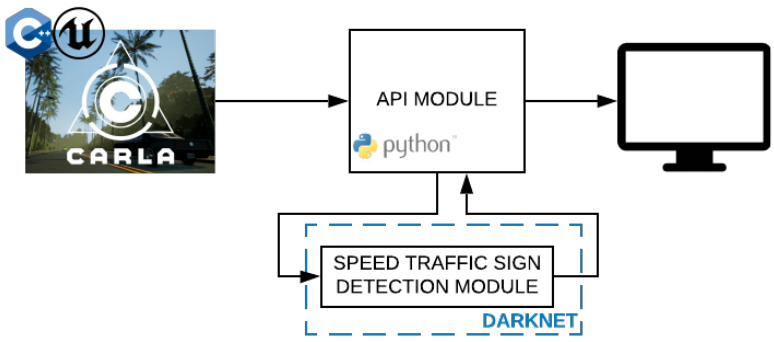

Figure 3: Project's Pipeline

Figure 3 shows the pipeline of the project and the main agents that take part in it: Darknet, the Speed Traffic Sign Detection Module (Yolo) and the CARLA simulator. Darknet is an open source neural network framework where the real-time object detection system Yolo is deployed [21]. This system is trained to recognise the speed traffic signs of the CARLA simulator and then, the trained model is integrated to it. After modifying some of the CARLA simulator source code, the trained Yolo model is able to detect the maximum allowed speed of a road and take

some decisions accordingly. Therefore, the following sections are going to explain these main parts of the project and the processes which were followed in each step.

2.1 Darknet and Yolo

As it was previously explained, Yolo is a state-of-the-art real-time object detection system that in comparison to other detectors, is extremely fast and accurate. This system is built on Darknet [21], an open source neural network framework that supports both the Central Process Unit (CPU) and the Graphic Process Unit (GPU) and that is written in C and CUDA. In order to get a system reproducible in most of the environments and with a low computational cost, it has been implemented on the CPU. For this, the Darknet installation process was computed following the documentation posted at its creator's website [21]. In Chapter 3, some comparisons between running the system on the CPU and on the GPU are performed because, as the Darknet's creator says: 'Darknet on the CPU is fast but it's like 500 times faster on GPU!' [21].

The next step was to train the network to detect our custom objects, in that case, CARLA's road speed signs. For this, we used the tiny model of the third Yolo version (Yolov3-tiny), i.e. the latest and fastest release. Yolo is a real-time object detection system based on regression, that predicts the classes and the bounding boxes for the whole image in one run of the algorithm. Its network design is inspired by the GoogLeNet model for image classification [16] but instead of having 22 deep Convolutional Neural Networks (CNN) layers, the tiniest version of the third model has 16 layers and the full third model version has 75 layers, both followed by 2 fully connected layers. This is the main difference between both models, what implies that the first one is faster than the second one, however, it is less accurate. Since the system was run on the CPU, we decided to use the fastest model minimizing the additional loading time introduced when integrating our module within the simulator engine. Moreover, the tiny model works better than the full model for constrained environments and to detect both small and large objects except for detecting small objects that appear in groups [22]. In the cases faced in this project, the speed

traffic signs change their sizes while the vehicle is getting closer to them, so this characteristic of the network fits perfectly to our needs. Also, they do not appear in group. However, the network has to be configured properly in order to train it correctly and improve by this way its accuracy. The changes that were made to the parameters that configure the model are explained in the following section: Network modifications.

2.1.1 Network modifications

First, the configuration of the net was adjusted to the case we need to face. The first step was changing the batch size from 1 to 64. This variable defines the number of samples (images) that will be propagated through the network at each iteration. Choosing a batch size number smaller than the total amount of training images has some advantages: less memory space is required during the training procedure because the network is trained with fewer samples and it is also trained faster because the weights are updated after each propagation. On the other hand, the smaller the batch variable is, the less accurate the estimation of the gradient will be. By default, a stochastic batch (batch size equal to 1) is set but a batch size of 64 deals better with the Yolov3-tiny architecture network [21]. Also, the number of maximum iterations was changed: the *max_batches* variable is set multiplying 2000 by the number of classes. That is because it is considered that 2000 iterations per class are needed to train the network properly. In our case there are 3 classes, as it will be explained below in section 2.2, so 6000 iterations will be needed in order to train the model. The learning rate is left by default: 0.001. After the 80% and the 90% of the maximum iterations value the learning rate will be adjusted, so the *steps* variable is set as '4800, 5400'. Finally, two more changes were performed. First, the network-resolution was increased from 412x412 to 608x608 in order to increase the precision and to allow the net to detect small objects. And secondly, a *flip* variable for disabling data augmentation in terms of flipping was added. Usually, flipping the training samples is used in deep learning training procedures in order to gain more variability over the sample images but, in this case, it could be misleading because if the 90km/h road speed sign is flipped and rotated it can be interpreted

as a 60km/h traffic sign, and vice-versa. Table 1 shows the adjusted parameters for the configuration of the net at the training process.

Training Network Configuration			
batch	64	**exposure**	1.5
subdivisions	16	**hue**	0.1
width	608	**flip**	0
height	608	**learning_rate**	0.001
channels	3	**burn_in**	1000
momentum	0.9	**max_batches**	6000
decay	0.0005	**policy**	steps
angle	0	**steps**	4800, 5400
saturation	1.5	**scales**	0.1, 0.1

Table 1: Configuration values of the network

Regarding the architecture of the Yolov3-tiny network, it looks like a normal CNN that consists of a total of 16 convolutional layers and 6 maxpool layers followed by 2 fully connected layers at the end. Moreover, 3 Yolo layers were inserted between the following convolutional layers: 10 th and the 11 th, 13 th and 14 th and, finally, after the 16 th layer. All of the layers use a leaky ReLU activation function except the convolutional layer previous to the Yolo layer that uses a linear function. In order to properly train the network considering the input data set, some changes were performed. First, in each of the 3 Yolo layers, the number of classes was changed to the number of objects learned by the model: 3. Also, at each convolutional layer before each Yolo layer the number of filters was modified following this rule: filters = (classes+5)×3. Finally, *random* value was set to 1 in order to increase the Yolo precision by training it at different image resolutions, sizes and allowing generalization over different object sizes.

Once the model was ready, the training and validation data set were prepared. For this, we used one of the CARLA Simulator's functionalities, saving one out of three frames on 10 sessions[1] of 6000 frames each, where we mixed different weather condition scenes in different scenarios. Then, we selected and filtered these frames in order to have a relevant amount of images of each object class and we labelled

[1]We refer to sessions as the continuous recording of simulated driving during a predefined amount of time, in this case 400 seconds.

them. To perform the training procedure, the same amount of images with and without labelled objects were added to both data sets in order to make the most robust model to False-Positive (FP) images. This amount was exactly 3,776 images for the training data set and 944 images for the validation data set, but Table 2 exposes the total number of images extracted from each CARLA session detailing some information such as which was the weather condition at that session or how many of these images were labelled. In addition, in order to contribute to the self-driving research, the whole filtered data set with more than 30,000 frames can be found at the Git Hub site[2] prepared for this project, where can be freely downloaded. The scenarios of the CARLA version used for this project have only 3 types of road speed signs: 30km/h, 60km/h and 90km/h, so therefore we have also considered these 3 classes in this thesis.

Data obtained from CARLA						
Session	Weather Condition	Total Frames	Labelled Frames	30 km/h	60 km/h	90 km/h
1 (Town01)	Sunny	4,907	144	58	7	79
2 (Town01)	Sunny	4,474	333	162	64	107
3 (Town01)	Cloudy	3,783	202	57	51	94
4 (Town01)	Sunny	4,289	268	109	36	123
5 (Town01)	Rainy	4,317	163	82	22	59
6 (Town01)	Cloudy	4,084	99	69	4	26
7 (Town01)	Rainy	3,781	190	91	26	73
8 (Town02)	Cloudy	529	529	296	189	44
9 (Town02)	Sunny	318	318	182	102	34
10 (Town01)	Sunny	114	114	0	46	68
TOTAL		30,872	2,360	1,106	547	707

Table 2: Data extracted from CARLA

2.1.2 Training Procedure

In order to avoid training the Yolov3-tiny network on the CPU (that is extremely slow) we used the free Jupyter Notebook environment from Google called Colaboratory (or Google Colab). This service allows executing some code and processing it at the cloud allowing the access to very powerful resources: such as the free use of the Tesla K80 GPU with a total RAM of 12 GB and more than 350 GB of storage. Once a procedure is started it can be used up to 12 hours consecutively, which is more than enough for this project.

[2]https://github.com/martisaju/CARLA-Speed-Traffic-Sign-Detection-Using-Yolo

To perform the training and the validation procedure properly, we prepared the environment on the Google Colab notebook leaving it as if it were our local Darknet folder. Then we just executed the Darknet training command using the Yolov3-tiny model previously designed and the data sets previously built. Stopping the training process trying to avoid the underfitting or overfitting phenomena, is not a trivial task. Darknet's site details that 2000 iterations per class are usually enough, but the training average loss and if it does not longer decrease between consecutive iterations should also be considered. It is also remarked that it is important to get the weights at the Early Stopping Point and in order to choose them the last weights files are compared. This comparison is performed by evaluating the validation set previously uploaded and getting the weights file that is giving the smallest error function value, the highest mean Average Precision (mAP[3]) and the highest Intersection over Union (IoU[4]) value. Table 3 shows the results of the evaluations of the 6 created weights files (one after each 1000 iterations) after 8 hours of training.

Weights files comparison			
Weights Files	Avg Loss	maP	IoU
1000 iterations	0.2612	38.11%	28.25%
2000 iterations	0.1559	82.80%	58.22%
3000 iterations	0.1013	89.49%	66.26%
4000 iterations	0.0618	90.93%	72.18%
5000 iterations	0.0482	91.62%	75.39%
6000 iterations	0.0521	91.56%	75.36%

Table 3: Table comparing the obtained weights from the training procedure

Taking the results shown in Table 3, the weights file corresponding to the 5000 th iteration has the highest maP and IoU values and smallest Avg Loss value. These weights are the chosen ones to perform the detection process in CARLA's simulator. In Chapter 3, all these weights will be tested on a test data set analyzing their performance and other metrics such as: the precision, the recall and the F1-Score. Moreover, True Positives (TP), False Positives (FP) and False Negatives (FN) rates will also be discussed.

[3]mAP is the mean of the average value of the precision across all recall values. It is a popular metric to measure the accuracy of the object-detection algorithms.

[4]IoU is an evaluation metric of object detectors that measures the overlap ratio between the ground truth and the predicted boundaries.

2.2 CARLA simulator

CARLA is an open-source simulator for autonomous driving research built on the Unreal Engine that has flexible APIs, where the users can change and control some aspects of the simulation. This simulation is performed on realistic urban layouts from where the users can get some inputs from the environment and take decisions accordingly. As CARLA deals with a large community of users and researchers, it is constantly in development. Since 2017 many versions (34 and upgrading) have been released and, as Figure 4 shows, a roadmap until December 2019 exists, meaning that some improvements will be done in the future.

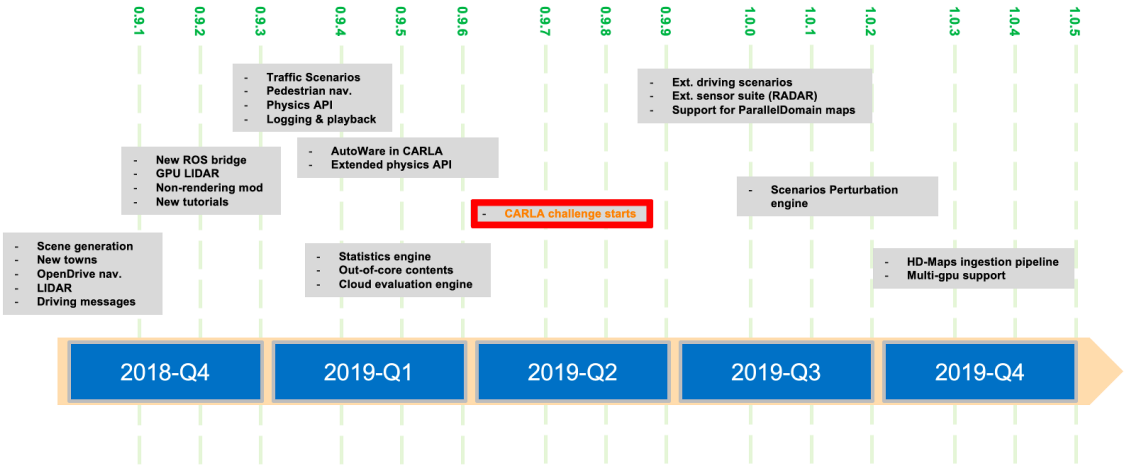

Figure 4: CARLA timeline [23]

Unfortunately, the compiled and stable version of CARLA for Windows (CARLA 0.8.2) is not the latest version deployed but for the aims of this project, it gathers the required features and characteristics. The compiled version of CARLA 0.8.2 has two available scenarios: Town01 and Town02, and many other parameters that can be set such as weather conditions, number of vehicles or pedestrians, among others. As it was previously explained in the 'Darknet and Yolo' section, three weather conditions were considered for this project: sunny, cloudy and rainy scenes, but CARLA allows the user to choose between 15 weather condition presets. For this project it is relevant to have the chance to change the environment conditions because the type of light, its angle of incidence and the amount of light that impacts

to the objects are different depending on the weather. Regarding the speed traffic signs that appear in CARLA 0.8.2 version, in both scenarios there are only three types of road-signs: 30km/h, 60km/h and 90km/h. As both maps are set in urban scenarios, the frequency of appearance of these three road signs is not uniform. Checking again the values in Table 2 it can be seen that the speed traffic sign that appears the most is the 30km/h speed-limit-road-sign (about 50% of the times that a road sign appears). The other two signs appear in a more uniform frequency among them: 25% each.

This simulator, that is built on a C++ engine, can be controlled with an external client script that can control most of the aspects of the simulation. The Python API module provided by the researchers contains a Graphical User Interface (GUI) implemented with PyGame where the user can control his or her vehicle manually. Moreover, some sensors such as 'light imaging detection and ranging systems' (LIDARs), RGB or Depth Map cameras can be added to the vehicle in order to get information about the environment and retrieve these measurements to take some decisions during the driving process.

Regarding this project, a camera was added to the vehicle on the frontal part of the car aiming to simulate the driver's point of view. This camera gets the RGB information of the scene every 5 frames (the render works at 15 fps) and saves it leaving it ready to be evaluated by the model and overwriting the previous saved image. Figure 5 shows a scene captured by the camera attached to the vehicle where a 90km/h speed traffic sign appears.

To run the trained Yolo model on CARLA and detect if a speed traffic sign is on the scene and in that case recognise it, a function called *_traffic_sign_recogniser* was implemented. In that function, every 5 frames the previously saved picture of that current scene is read and evaluated by the model. At that point a short delay time (around 130 ms) is induced due to the CPU Darknet framework execution, as explained in the first section of the methodology. Then, an output file is generated indicating whether a speed traffic sign has been detected, which sign it is and its location. If no speed road sign has been detected, an empty file is saved. If the saved

Figure 5: Scene recorded by an RGB camera attached to the vehicle

file is not empty, ergo a speed traffic sign has been detected, it is sent to the render function that prints on the top of the screen the maximum allowed speed of the road and stays embedded until a new speed limit road sign is detected. Moreover, this detected speed limit sign is used in two different ways depending on the flag previously set by the user on the execution of the simulator. This will be explained in the Applications section.

Figure 6 shows an example of an embedded 90km/h speed road sign that has been detected. Also, the current speed of the vehicle is printed in the down-right part of the screen.

Appendix A includes some screenshots of the chronological modifications done in CARLA's *manual_control.py* script to integrate and deploy the recognition system and its applications. Moreover, at the Git Hub[5] prepared for this project it can

[5]https://github.com/martisaju/CARLA-Speed-Traffic-Sign-Detection-Using-Yolo

Figure 6: Speed Traffic Sign detected embedded on the screen

be found all the needed material to replicate this project in another machine: both the training, validation and testing data sets, the configured Yolo network and its weights and the modified *manual_control.py* CARLA's script.

2.3 Applications

In the objectives section of this thesis, it is exposed that the main motivation that leads to this project is to train a system able to detect and recognise speed traffic signs in CARLA and then take decisions accordingly. Many decisions can be taken once a vehicle has detected the maximum allowed speed on a road but for this project two example applications have been deployed. In the introduction it was explained that most of the fatal car accidents on roads are due to exceeding their maximum allowed speed. Warning the user to reduce the vehicle speed or automatically reducing its speed could help to decrease the number of tragedies in our

roads. To select one of these applications, the user has to turn on the application flag (*-app*) and choose between two options: 'Warning' or 'Control' in the execution command.

2.3.1 The Warning Application

If the user has set the 'Warning' flag at the execution command of the simulator (*-app Warning*), once a speed-road-limit has been detected it is checked if the current speed of the vehicle exceeds it. If the vehicle speed is lower than the detected one, the screen reminds the basic information: the detected speed traffic sign at the top of the screen and the current speed of the vehicle, painted in green, at the right-bottom-corner (Figure 6).

Figure 7: Screenshot showing the message printed by the Warning-Application

On the other hand, if the current speed of the vehicle exceeds the detected one, a warning message appears at the bottom part of the screen. This message says:

'REDUCE YOUR SPEED' in red, advising the driver to reduce his or her vehicle speed because it is exceeding the maximum allowed speed on that road. Also, the current speed of the vehicle changes its colour from green to red. Figure 7 shows an example of the previous explanation, showing the warning message notifying the user that his or her vehicle speed (34.04km/h) exceeds the maximum allowed speed on the road where he or she is driving.

2.3.2 The Control Application

On the other hand, if the user has selected the 'Control' mode (*-app Control*), then the system is more restrictive. The same happens for the 'Warning' mode: if the vehicle speed is lower than the maximum allowed speed on that road, the screen shows the basic information. However, in that case, if the current speed of the vehicle exceeds the maximum allowed speed detected plus the 10%[6] of that speed, then the vehicle reduces its speed automatically setting it below the allowed threshold. To notify the user that the speed of the vehicle has been reduced, a message in red colour saying: 'SPEED REDUCED' is printed at the bottom part of the screen, as it is shown in Figure 8.

Regarding how the vehicle speed is reduced using the CARLA Python API once it is realised that the vehicle exceeds the maximum allowed speed-road-limit (plus the 10% of this limit), a message is sent to the function that controls the keyboard pressed keys (*_get_keyboard_control*) notifying this detected abnormality. This function checks which key is pressed on the keyboard at each iteration (i.e. every 50 ms), reading, in this way what the vehicle has to do in the following iteration: turn left/right, brake or accelerate. Once this function gets this message, the throttle is turned off and the break is activated reducing the current speed of the vehicle and fitting to the speed-road-limit. In Appendix A, Figure 16 shows the modifications done in the *_get_keyboard_control* function to decrease the vehicle speed for the next loop-iteration.

[6] In Spain a vehicle can exceed by a 10% of the maximum allowed speed-limit on a road if it is overtaking another vehicle.

Figure 8: Screenshot showing the message printed by the Control-Application

Chapter 3

Experiments and Results

In the third Chapter of this thesis are going to be exposed some of the experiments and trials done during this project. In addition, two techniques that were initially tested to detect the CARLA traffic speed signs will be explained. Moreover, in the Results section, there will be discussed the results that were obtained from the training phase of the model and also a comparative analysis evaluating the obtained results on running the system both on the CPU and on the GPU will be performed.

3.1 Experiments

Before employing an object detection algorithm to recognise the CARLA traffic speed signs, a simpler algorithm to detect the shape of these signs was thought to be used. Then, a digit recogniser algorithm would be applied to get the detected traffic sign information. The shape that usually prevails in the speed road signs is circular, so we firstly thought about employing an algorithm able to detect circles. For this, an algorithm based on the Hough Transform technique would be used. The Hough Transform is a technique that can get the features of a particular shape within an image. It was initially designed for identifying straight lines in an image, but a specialization of this technique was developed for detecting curve lines allowing it to identify circles and ellipsis.

Figure 9: Traffic Speed Sign detected by the Hough Circle Transform algorithm

Firstly, an OpenCV module to detect the circular objects that appear in the CARLA scenes by using the OpenCV Hough Transform function was integrated in the *manual_control.py* script. A minimum and maximum radius value of the circles that would be detected must be set in that function, in order to get the most accurate search. The minimum radius was defined to 5 pixels, while the maximum to 40 pixels. This large difference between both radiuses is justified because the traffic signs tend to increase their size while the vehicle is getting closer to them, so they comprise a wide range of sizes. Also, a minimum distance between the centers of the detected circles was defined, assuming that with a small distance number multiple neighbor circles may be falsely detected. Since usually the traffic speed signs appear individually, that distance was set to 1000 pixels.

When this was tested in CARLA, it worked well and really fast (around 23 ms) and it could detect road speed signs when they appeared on the scene, as Figure 9 shows, but many False Positives traffic signs were also detected. As the radius range values were too distant between them, many quotidian objects with circular shapes were actually detected, such as: umbrellas, clouds, circular car brands, wheels... Figure 10 shows two images: at the top image, an all-terrain vehicle driving in front of the

agent vehicle that has a spare-wheel at the trunk can be seen. In the picture shown below, this wheel has been detected as a speed traffic sign due to their similarities in terms of shape and size.

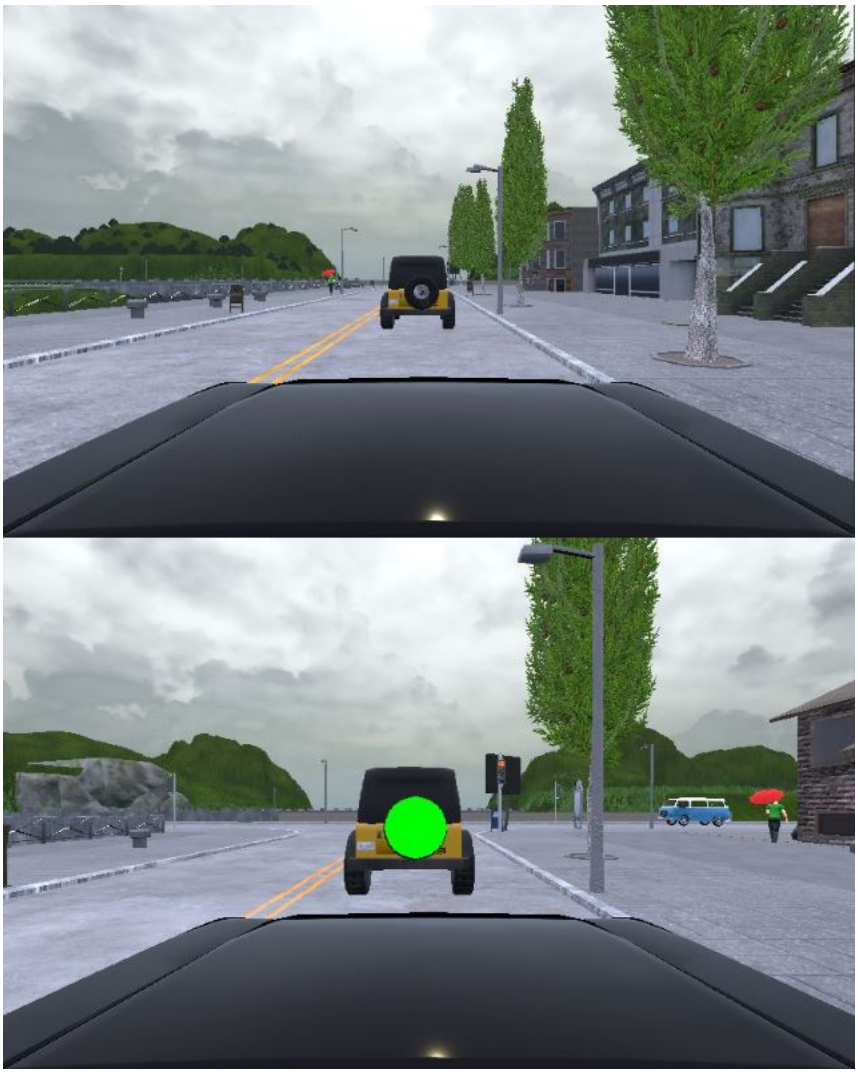

Figure 10: Circle Detection test

After testing the algorithm in different radius sizes and different weather conditions, it was rejected because its False Positives rate was too high. Then, the power, the robustness and the ability to generalize that the deep neural networks algorithms provide was considered in order to recognise the speed traffic signs. After building our own model, a pretrained network that recognized traffic signs from natural images by using a widely known data set was tested: the German Traffic Sign Dataset (GTSD). The architecture of this model was based on the LeNet CNN,

a network that has a straightforward architecture, which is widely used in many models to recognise handwritten digits. Testing this pretrained model on the GTSD, provided acceptable accuracy results (see Table 4), but we realized that it was trained on cropped images so that indicated that we should first try to apply a speed traffic sign detection procedure to our images.

Training Set Accuracy	99.5
Validation Set Accuracy	96.8
Test Set Accuracy	94.6

Table 4: Results on the pretrained LeNet model

The purpose of detection is to find the locations and sizes of the existing traffic signs within an image [8] to then apply a classification algorithm and assign a class label to each of the detected traffic signs. To detect the traffic signs, sliding window based methods, such as the Aggregate Channel Features (ACF) or the Integral Channel Features (ICF) [24], and regions of interest (ROIs) based methods are commonly used to apply the detection procedure in delimited zones within the image, in which traffic sign candidates are then found. In our case, we first considered placing these ROIs for detecting the traffic signs on the right-side of the road in the image and then defining a 3-D search box as it is done in project [25]. For this, we firstly decided to detect the lane lines of the road. These lines delimit the width of each lane using colours that contrast with the gray colour of the asphalt, such yellow or white. Usually, these lines are placed both in the middle of the road (to separate the lanes), and on each side of the road to delimit the road with the ditch. CARLA scenarios are set in two urban environments, so in both cases, there are sidewalks that have a colour similar to the one of the road on each side of the road. Between the road and the sidewalk there is a delimiter whose colour is darker than the sidewalk and the road. We tried to identify these lines (the yellow-middle line and the grey-right-sidewalk delimiter) by defining two colour masks, before denoising the image using a Gaussian blur filter. After that, the Canny edge detector was employed to determine the edges of that filtered image and finally, the Hough Transform algorithm was applied to detect the image lines. Detecting the yellow-middle line was not a challenging task due to its contrast with the road colour, but detecting

the sidewalk delimiter was not achieved with convincing results. Depending on the weather condition and the shadows that were projected over the road, this delimiter colour changed in a very different range of grey nuances making the detection task too demanding, as Figure 11 shows.

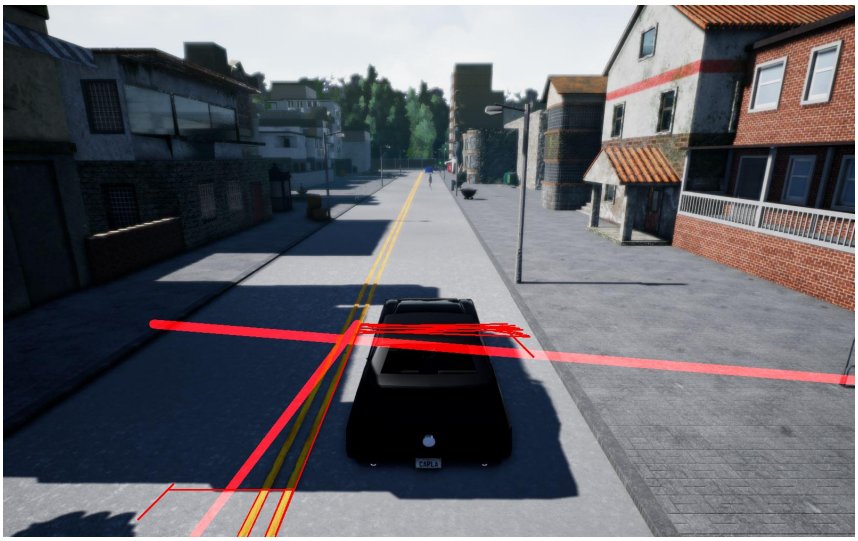

Figure 11: Lane Lines Detection test

At that point, we realized that defining ROIs over the CARLA scenes for detecting and identifying the speed traffic signs would be a challenging task. Moreover, evaluating the whole image with different sliding windows sizes over several locations would consume many computational resources and spend too much time. After some research, we decided to employ Yolo that, as it was explained in Chapter 1, it perfectly fits with all the requirements specified in this project. As previously explained, Yolo sees the entire image during the training and testing phases encoding contextual information about the object classes as well as their appearances instead of using a sliding window over several locations in an image. Also, it generalises extremely well, and the fact that our case faces with different weather conditions that change the type and amount of light that impacts to the objects, this is something that has to be considered.

The following section will analyze some metrics on the weights obtained in the training phase of the network and the comparison between running the system on the CPU or on the GPU.

3.2 Results

In order to test the trained network, a test set of 1,180 images extracted from CARLA in a single session was created. As in the training set, the same amount of images with and without labelled objects were added to the test set (see Table 5).

Appearance of each class in the test data set				
Unlabelled	**Labelled**	30 km/h	60 km/h	90 km/h
590	**590**	223	193	174

Table 5: Test set in detail

Three more metrics, apart from the mAP and the IoU, were evaluated to choose the weights file that would be used within the system. These metrics were the precision, the recall and the F1-Score. The precision measure is the ratio of correctly predicted positive observations to the total predicted positive observations. The recall metric is the proportion of how many of the total amount of positives observations have been already correctly predicted as positive. Finally, the F1-Score is a measure used to seek a balance between the precision and the recall, in other words it is the weighted average of these metrics. To get these measures, three rates must be obtained: True Positives[1] (TP), False Positive[2] (FP) and False Negative[3] (FN) rates. Moreover, the percentage of cases per class that were correctly-predicted were studied in each of the generated weights files. Table 6 and 7 expose respectively these results.

Results of testing the weights files (1)						
Weights Files	Precision	Recall	F1-Score	TP	FP	FN
1000 iterations	0.39	0.33	0.36	194	298	396
2000 iterations	0.77	0.82	0.80	484	144	106
3000 iterations	0.87	0.91	0.89	538	41	52
4000 iterations	0.90	0.93	0.92	548	58	42
5000 iterations	0.92	0.94	0.93	550	46	40
6000 iterations	0.91	0.94	0.92	550	52	40

Table 6: Metrics obtained with the test data set

The data from above shows the results from the metrics which have been previously commented. This results were obtained from the weights files produced during the

[1] Amount of positive observations that were actually predicted as positive.
[2] Amount of negative observations that were falsely predicted as positive.
[3] Amount of positive observations that were falsely predicted as negative.

Results of testing the weights files (2)			
Weights Files	30 km/h	60 km/h	90 km/h
1000 iterations	41.91%	34.43%	37.99%
2000 iterations	73.38%	81.27%	93.76%
3000 iterations	83.82%	85.16%	99.47%
4000 iterations	85.40%	87.64%	99.74%
5000 iterations	86.89%	88.09%	100.00%
6000 iterations	86.79%	87.78%	100.00%

Table 7: Performance values on the test phase

training phase. Analyzing these results and the ones obtained in the validation test (Table 3), we can see how the values regarding the weights file on the 5000 th iteration are the highest ones. This justifies that this were the ones chosen to be integrated in the object-detection system. The fact that the files with the highest weights do not refer to the ones obtained in the last iterations indicates that an overfitting phenomena occurs between iterations 5000 and 6000.

Even though the project was initially thought to be deployed on the CPU, its integration to CARLA was tested on both the GPU and the CPU. The differences between executing the process in both environments are notable. Running the system on the CPU implies that the resolution in which CARLA is executed needs to be smaller than when it is done on the GPU. In this way, CARLA's low resolution flag is activated simplifying the render of the graphics (Figure 12). This allows CARLA's render to run in 15 fps, allowing that the execution of Darknet on the CPU not to be too computationally expensive and evaluating the current CARLA's scene in 130 ms. This result is far from being a real-time evaluation but as the Darknet's creator mentions in [21] it runs faster on the GPU. Moreover, it has to be noted that the CPU has to process both the Unreal Engine running the CARLA simulator and the Darknet framework running the Yolo's evaluation.

A NVIDIA GeForce GTX 1070 GPU was used in order to test the system on the GPU. The obtained results (regarding the execution time and quality) noticeably improved. Running CARLA and the speed traffic sign recognition system on the GPU allowed us to choose CARLA's high quality render option. This option maintains the original CARLA graphics that are expected to be as realistic as possible

(see Figure 13). At that mode the render works 3 times faster than in the low quality mode, at 45 fps. Once the whole system was tested on the GPU, it surprisingly improved by 16 times the execution time spent on the CPU test. This system is able to evaluate the current CARLA scene, determine if any speed traffic sign exists and recognise its identity in a very challenging real time, 8 ms.

Figure 12: CARLA scene in low quality mode

Figure 13: CARLA scene in high quality mode

Chapter 4

Discussion and Conclusions

Even though many project studies regarding the speed traffic sign recognition process have been done previously obtaining very satisfactory results, none of these was integrated on CARLA. This simulator is supported by a large community of users and researchers who constantly contribute to the development of this platform but, in spite of this no CARLA labelled images data set exist. Performing this project, a real-time CNN was trained in order to detect and recognise the CARLA speed traffic signs. For this, three data sets were created to train, validate and test the network. These, that can be freely downloaded at the Git Hub site prepared for this project among other images, consist of 5,900 images. These images were split as follows: 3,776 images for the training set, 944 images for the validation set and 1180 images to build the test set. Each of these sets contained the same amount of labelled and unlabelled images, with the purpose to made the model as robust as possible to False Positive cases. Previously to the training procedure, the Yolo CNN was configured taking into account the type of objects that would be detected (in terms of size, colour, shape...) and the contextual cases where it would be tested. In our case, getting a model robust to the environmental changes was a challenging task because depending on the weather condition of each session, the amount of light and its angle of incidence over the objects changed. After configuring the network properly, the model was trained in the Google Colab platform, producing six files

of weights that were compared on the validation set. At this comparison, the file that got the smallest error function value and the highest mAP and IoU values was selected. To evaluate the performance of the trained network with the chosen file, it was tested on the test set getting satisfactory results about the Precision, the Recall and the F1-Score metrics, and making noticeable that some overfitting was induced in the training phase between the 5000 th and the 6000 th iterations. After that, an RGB camera sensor was added to the CARLA vehicle getting the environment information after each 5 frames and leaving it ready to be evaluated by the trained model. This model was also integrated to the CARLA code taking advantage from the CARLA Python API module and being able to determine in 130 ms on the CPU and in 8 ms on the GPU whether it exists or not a speed traffic sign within the CARLA scene. Moreover, two example applications were implemented. First, a non-intrusive application that notifies the user if the vehicle speed exceeds the maximum allowed speed of that road by printing on the screen a warning message. Secondly, an intrusive application was implemented breaking the vehicle if it overtakes the speed limit of the road plus its 10%, leaving the vehicle driving below the maximum allowed speed of the road.

Analyzing the performance and the overall results obtained in the project, some conclusions can be extracted. First, the initial research purpose was accomplished: to integrate a system able to detect and recognise speed traffic signs on a vehicle without any previously intelligent speed assistance device. In addition, two applications motivated by this detection procedure have been implemented, giving the system a more cognitive and perceptual utility. Moreover, this project highly contributes to the self-driving research, and both the created data sets, the trained model and the modified CARLA script can be found at Git Hub site[1] created for this project, allowing everyone to use them freely for his or her research. Finally, although the achieved results provide well-accepted performance values, as it is widely known for the deep learning algorithms, as many images are used for training the network, better weights are obtained on the training phase and better performance results are then achieved on the testing phase.

[1]https://github.com/martisaju/CARLA-Speed-Traffic-Sign-Detection-Using-Yolo

The performance of this project is encouraging and has implications for further research on self-driving studies. Particularly, implications in order to make research on the speed traffic sign recognition field and the applications that can be deployed to help the driver with the purpose to make the driving process easier and safer.

List of Figures

List of Tables

Bibliography

[1] European Commission. Safer roads for all: The eu good practice guide. URL https://ec.europa.eu/transport/road_safety/sites/roadsafety/files/pdf/safer_roads4all.pdf.

[2] Bimbraw, K. Autonomous cars: Past, present and future a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology. In *2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, vol. 1, 191–198 (IEEE, 2015).

[3] European Union Road Federation. Improved signage for better roads. URL http://www.erf.be/wp-content/uploads/2018/01/ERF_Position_Paper_on_Vertical_Signage_Final_7.pdf.

[4] European Commission. Speeding: more and more severe accidents. URL https://ec.europa.eu/transport/road_safety/specialist/knowledge/speed_en.

[5] The European Commission. Road safety awards (2017). URL https://ec.europa.eu/transport/media/news/2017-06-26-road-safety-awards_en.

[6] HERACLES Group. Good driver project (2017). URL http://www.kalosodigos.gr.

[7] Wikipedia. Adaptive cruise control (2019). URL https://en.wikipedia.org/wiki/Adaptive_cruise_control. Online; accessed 27-March-2019.

[8] Luo, H., Yang, Y., Tong, B., Wu, F. & Fan, B. Traffic Sign Recognition Using a Multi-Task Convolutional Neural Network. *IEEE Transactions on Intelligent Transportation Systems* **19**, 1100–1111 (2018).

[9] Vennelakanti, A. *et al.* Traffic sign detection and recognition using a cnn ensemble. In *2019 IEEE International Conference on Consumer Electronics (ICCE)*, 1–4 (IEEE, 2019).

[10] Lowe, D. G. *et al.* Object recognition from local scale-invariant features. In *iccv*, vol. 99, 1150–1157 (1999).

[11] Dalal, N. & Triggs, B. Histograms of oriented gradients for human detection. In *international Conference on computer vision & Pattern Recognition (CVPR'05)*, vol. 1, 886–893 (IEEE Computer Society, 2005).

[12] Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection (2016). URL `http://arxiv.org/abs/1506.02640`. `1506.02640`.

[13] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A. & Koltun, V. CARLA: An Open Urban Driving Simulator 1–16 (2017). URL `http://arxiv.org/abs/1711.03938`. `1711.03938`.

[14] Shah, S., Dey, D., Lovett, C. & Kapoor, A. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In Hutter, M. & Siegwart, R. (eds.) *Field and Service Robotics*, 621–635 (Springer International Publishing, Cham, 2018).

[15] Girshick, R., Donahue, J., Darrell, T. & Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 580–587 (2014).

[16] Szegedy, C. *et al.* Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015).

[17] Deng, J. *et al.* Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255 (Ieee, 2009).

[18] Cordts, M. *et al.* The cityscapes dataset. In *CVPR Workshop on the Future of Datasets in Vision*, vol. 2 (2015).

[19] Houben, S., Stallkamp, J., Salmen, J., Schlipsing, M. & Igel, C. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks (submitted)* (2013).

[20] Wymann, B. *et al.* Torcs, the open racing car simulator. *Software available at http://torcs. sourceforge. net* **4** (2000).

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

[22] Redmon, J. & Farhadi, A. Yolov3: An incremental improvement. *arXiv* (2018).

[23] CARLA Researchers. Carla roadmap (2018). URL `https://github.com/carla-simulator/carla/blob/master/Docs/img/carla_timeline.png`.

[24] Dollár, P., Appel, R., Belongie, S. & Perona, P. Fast feature pyramids for object detection. *IEEE transactions on pattern analysis and machine intelligence* **36**, 1532–1545 (2014).

[25] Greenhalgh, J. & Mirmehdi, M. Recognizing text-based traffic signs. *IEEE Transactions on Intelligent Transportation Systems* **16**, 1360–1369 (2015).

Appendix A

CARLA Code

A.1 Speed Traffic Sign Detection Function

In order to perform the CARLA scene evaluation in an acceptable time, the Darknet framework with the calculated weights file and the Yolo network within it has to be executed and be ready to process an image at every moment. For this purpose, this framework is executed at the beginning of the session (see Figure 14) and an instance of this process is assigned into a global Python variable. Then, once an image has to be evaluated, it just has to be sent into that instance that is ready to analyze it. Figure 15 shows the function that has been implemented in the *manual_ control.py* CARLA's script, which gets this running process and sends to it the path of the image that has to be evaluated.

```
def darknet(message):
    os.chdir("C:/Users/Marti/Desktop/DARKNET-V2/darknet-master/build/darknet/x64")
    process = popen_spawn.PopenSpawn('darknet_no_gpu.exe detector test data/obj.data \
                                     yolov3-tiny-obj.cfg yolov3-tiny-obj_5000.weights \
                                     -dont_show -ext_output -save_labels') #Running Darknet
    print(message)
    return process
message = 'Darknet Started'
darknet_process = darknet(message)
```

Figure 14: Screenshot of the *darknet* function where it is launched the Yolo network to evaluate each incoming image

```python
def _traffic_sign_recogniser(self, new, detected_speed_loop):
    global darknet_process
    carla_scene_exists = os.path.isfile("Y:/Temp/carla_scene.jpg")
    if carla_scene_exists:
        carla_scene = (b"Y:/Temp/carla_scene.jpg")
        darknet_process.send(carla_scene+b'\n')

        labels_file_exists = os.path.isfile("Y:/Temp/carla_scene.txt")
        if labels_file_exists:
            labels_file = open("Y:/Temp/carla_scene.txt","r")
            predicted_labels = labels_file.read()
            if os.stat("Y:/Temp/carla_scene.txt").st_size != 0:
                #Check if any speed traffic sign has been detected
                sign_tsr = predicted_labels[0]
                if int(sign_tsr) == 0:
                    detected_speed_loop = '30'
                elif int(sign_tsr) == 1:
                    detected_speed_loop = '60'
                elif int(sign_tsr) == 2:
                    detected_speed_loop = '90'
                new = True

            else:
                detected_speed_loop = ''
    return detected_speed_loop, new
```

Figure 15: Screenshot of the *_traffic_sign_recogniser* function implemented in the *manual_control.py* CARLA's script

```python
def _get_keyboard_control(self, keys, args):
    """
    Return a VehicleControl message based on the pressed keys. Return None
    if a new episode was requested.
    """
    global flg_warning

    if keys[K_r]:
        return None
    control = VehicleControl()
    if keys[K_LEFT] or keys[K_a]:
        control.steer = -1.0
    if keys[K_RIGHT] or keys[K_d]:
        control.steer = 1.0
    if keys[K_UP] or keys[K_w]:
        control.throttle = 1.0
    if keys[K_DOWN] or keys[K_s]:
        control.brake = 1.0
    if keys[K_SPACE]:
        control.hand_brake = True
    if keys[K_q]:
        self._is_on_reverse = not self._is_on_reverse
    if keys[K_p]:
        self._enable_autopilot = not self._enable_autopilot

    if args.app == 'Control':
        if flg_warning == -1:
            control.throttle = 0.0

    control.reverse = self._is_on_reverse
    return control
```

Figure 16: Screenshot of the modifications done in *_get_keyboard_control* function

A.2 Speed Control

The modifications done in the *_get_keyboard_control* function in order to decrease the current vehicle speed can be seen in Figure 16. That speed is progressively reduced if the current vehicle speed exceeds the maximum allowed speed of the road plus its 10%. This utility has been packaged at the Control-Application functionality.

A.3 Render Function

Figure 17 shows the *render* function. At that function it is first analyzed whether any application mode has been set or not. If any of the two applications modes is set and if the current frame is multiple of 5, it is evaluated by the *_traffic_sign_recogniser* function. That function determines if a speed traffic sign appears on the scene. If it appears, it is printed on the PyGame screen the detected speed limit. Moreover, depending on the application mode and on the current vehicle speed, more information can also be printed at the bottom part of the screen, such as a warning or a control notification message.

```
if args.app:
    new = False
    detected_speed_loop = ''

    if (frame % 5 ==0):
        a = Image.fromarray(array, 'RGB')
        a.save('Y:/Temp/carla_scene.jpg')
        detected_speed_loop, new = self._traffic_sign_recogniser(new, detected_speed_loop)

    if new:
        detected_speed = detected_speed_loop

    if flg_warning == -1:
        r = 255
        g = 0
        b = 0
    elif flg_warning == 1:
        r = 0
        g = 255
        b = 0
    else:
        r = 0
        g = 0
        b = 255

    basicfont = pygame.font.SysFont(None, 80)
    text_detected = basicfont.render(detected_speed, True, (0,0,255))
    textrec = text_detected.get_rect()
    textrec.top = surface.get_rect().top
    textrec.midtop = surface.get_rect().midtop
    surface.blit(text_detected, textrec)

    text_current = basicfont.render(current_speed+'km/h', True, (r,g,b))
    textrec = text_current.get_rect()
    textrec.top = surface.get_rect().top
    textrec.bottomright = surface.get_rect().bottomright
    surface.blit(text_current, textrec)

    if args.app == 'Warning':
        if flg_warning==-1:
            basicfont = pygame.font.SysFont(None, 60)
            text_warning = basicfont.render('REDUCE YOUR SPEED', True, (r,g,b))
            textrec = text_warning.get_rect()
            textrec.top = surface.get_rect().top
            textrec.bottomleft = surface.get_rect().bottomleft
            surface.blit(text_warning, textrec)

    if args.app == 'Control':
        if flg_warning==-1:
            basicfont = pygame.font.SysFont(None, 60)
            text_control = basicfont.render('SPEED REDUCED', True, (r,g,b))
            textrec = text_control.get_rect()
            textrec.top = surface.get_rect().top
            textrec.bottomleft = surface.get_rect().bottomleft
            surface.blit(text_control, textrec)
```

Figure 17: Modifications done in the *render* function to analyze one after 5 frames and to print messages on the screen