EPFL

Projet de Master RO
2021-2022
Section de Microtechnique

Conditional imitation learning with pyramid perception modules

Maël Wildi

14 January 2022

Supervisors: Dr. Arnoud Visser (UvA)
Dr. Alexandre Alahi (EPFL)

Project conducted at the *Universiteit van Amsterdam*

Abstract

In this thesis, we train a driving agent to predict waypoints directly from a forward-facing camera, following the work done in *Cheating by Segementation* [1], itself inspired by *Learning by Cheating* [2]. End-to-end learning lets the neural network find its own representation of the environment, without restraining it to human-readable features. It leads to models that are lighter and that could theoretically be closer to the optimal. From traffic lights to vehicles, the model should be able to perceive objects of various sizes in order to construct an accurate representation of the environment. The objective is to change the network architecture in order to reduce the number of infractions made by the agent.

We compare two approaches aiming at letting the network build a better representation. (i) Using a Pyramid Pooling Module [3], we bring global context into the feature maps. (ii) Using a Feature Pyramid Network [4], we fuse outputs coming from different layers to obtain semantically strong and spatially accurate features. Both agents demonstrate a lane keeping rate of more than 94% on the *NoCrash* [5] benchmark. The one relying on the Feature Pyramid Network manages to have a number of traffic light infractions per hour of 10.6 in test conditions, which is in the same range as state-of-the-art implementations *Learning by Cheating* and *Learning to drive from a world on rails* [6].

Acknowledgments

I would like to thank Dr. Arnoud Visser of the *Informatics Institute* at *Universiteit van Amsterdam* for giving me the opportunity to carry out this project. His guidance and advice have been really valuable. I would also like to thank Dr. Alexandre Alahi of the *Visual Intelligence for Transportation Laboratory* at *École Polytechnique Fédérale de Lausanne* for supervising me during this project. His comments and suggestions helped me define the problem statement and the experiments. Finally, I would like to thank the members of the *Intelligent Robotics Laboratory* for their welcome and their helpful ideas.

Contents

Chapter 1

Introduction

Driving assistance has taken a considerable spot in everyday driving since the introduction of the anti-lock blocking system (ABS) by Bosch in 1980. From rear sensors signaling obstacles to adaptive cruise control maintaining a constant speed, relying on these assistance systems has become common.

Since the mid 2010's, several manufacturers propose self-driving cars that technically enable their driver to let go of the vehicle commands, although for legal reasons the driver is still required to have their hands on the steering wheel. In 2021, Honda in Japan and Mercedes in Germany received the authorization to deploy a vehicle where the driver is allowed to let the car drive itself, as long as they do not exceed a speed of 60km/h and are able to take back control if needed.

Self-driving vehicles currently available in the industry are based on a modular approach, where perception, planning and control are separated from one another. In the past decade, the use of convolutional neural networks (CNN) has exploded, thanks to the increased power of computers. It is now widely used for perception tasks, as it enables to learn a model capable of recognizing objects from a large amount of annotated images.

Besides, it has opened the door to end-to-end learning, an alternative to the modular method [7]. In the case of autonomous cars, it consists in proposing control commands directly from a sensory input, letting the network learn the intermediate features. Thus, these learned features are not human-readable and there is no longer the need to define explicit rules for every situation the vehicle might encounter. There is also no need to build a model of the environment.

Imitation learning is, alongside the emerging reinforcement learning approach, the main method to perform end-to-end learning for autonomous cars. It consists in training a model to reproduce the actions an expert would have performed if presented to the same situation.

Many models rely on data collected by a human driver [8], however driving simulators such as Car Learning to Act (CARLA) [9] are more and more used as they are getting very realistic. Most importantly, a simulator allows to put the autopilot in challenging situations, thereby showing the neural network how to recover without putting anyone at risk. Secondly, it enables to have scenarios where other cars and pedestrians are disrespecting traffic rules, which is important in order to learn a robust policy.

Figure 1.1: Illustration of CARLA simulator, widely used for autonomous driving [5]

The sensory input consists in almost every implementation of at least a RGB camera, as it is the only sensor that enables to distinguish the traffic light state, while providing also enough information to the network about the environment. Some works use additional sensors such as depth camera [10] or LiDAR [11], relying for instance on transformers [12] to smartly fuse the different inputs together.

Nonetheless, [5] managed to have competitive results using only a RGB image from a driver point of view as input. This could allow to easily collect data using existing cars driven by everyday commuters, to perfect a model pre-trained on a simulator to the particular driving environment of a country.

Cheating by Segmentation (CBS) [1], modified *Learning by Cheating* (LBC) [2], one of the best performing driving policy on CARLA simulator, to rely only on images coming from a driver perspective instead of a bird's-eye view (BEV). It lead to promising results, however the number of collisions and traffic lights infractions increased.

Thus, this thesis pursues the goal of generating a robust end-to-end imitation learning driving policy using RGB and semantically segmented images from a driver perspective as only inputs. The focus is on reducing the number of traffic light infractions and collisions. In order to do this, we rely on pyramid perception modules that are described in the following section.

Chapter 2

Theory

In this section, the methods and theoretical concepts that are essential to this thesis are introduced.

2.1 Conditional imitation learning

Imitation learning is an end-to-end method, which means that, in the case of autonomous driving, it maps observations inputs directly into vehicle commands [5]. There is no longer separate modules for sub-tasks that are find in decomposed learning such as path planning or obstacle avoidance. This has the advantage of not constraining the model to human-readable features and rules, which are rarely exhaustive and not necessary for the computer. Besides, the dataset is easily collectable, as it consists only of sensory inputs and measured vehicle commands.

However, complex functions are required to map sensory inputs to vehicle commands, therefore the network must have a high capacity to learn a model capable of performing this mapping. In addition, it is much harder for its developers to examine the reasons that lead the model to a specific output, as there is no intermediate representation or classification made that could be used to check the correct behaviour of the model.

Imitation learning relies on demonstrations by a driver, which can be a person in the real-world or the autopilot of a simulator, referred as "expert". These demonstrations enable to construct a dataset made of sensory inputs (such as RGB image or LiDAR scan), vehicle commands (steering angle, throttle, brake) and vehicle state (position, orientation, speed). The network tries to learn a model which, given the same inputs as the expert, outputs similar vehicle commands and thereby learns to imitate the expert.

Conditional imitation learning (CIL) addresses an issue that is characteristic of imitation learning: given a same environment and its associated observations, an expert can take different actions from one run to another. This is often the case in driving when the car arrives at a crossing: there is no wrong direction to take, it depends only on the intentions of the driver. Therefore, for the network to learn correctly, it is important to include them in the model. Such intentions can take the form of a directional command, for instance "turn left" or "go straight". This allows to use multiple identical heads in the network, that are trained exclusively with data corresponding to their associated command. The resulting agent selects the action predicted by the head associated to the provided current intention of the driver.

2.2 Knowledge distillation

An important quality for a model is to be able to generalize well on never seen before data. This requires to be trained on large datasets and using a rather complex network, especially when performing end-to-end learning. Therefore, the resulting model is often too heavy to be run in real-time. There come into use knowledge distillation, often implemented in the form of a teacher-student network.

This method consists in training two models with different networks instead of one. First, a heavy network is trained offline on large datasets. It is called the "teacher". Once trained, it supervises a lighter network, the "student", on a common output. The resulting student model is the one that is actually deployed.

The teacher-student method can also be used to decompose the learning of a task into two stages. They must have a common output so that the teacher is able to supervise the student. In this approach, the student network is not necessarily lighter than the teacher.

2.3 Perception

2.3.1 Convolutional neural network

A convolutional neural network (CNN) is a neural network especially adapted for images. It takes as input a set of images, each expressed as a matrix of pixels, and assembled into a unique tensor of size *(number of images, height, width, number of channels)*. A CNN consists of a sequence of convolutional layers that have their own set of kernels. These kernels have the same number of channels as the input images but a smaller spatial dimension. Convolution is performed by sliding the kernel over the input. Each kernel produces a single-channel neuron output that represents the presence of a particular feature in the image. This process has the advantage of preserving the spatial dependencies in the image.

Then, it goes through an activation function, which role is to introduce non-linearities in these neurons outputs, thus making it possible for the network to learn more complex models. The output of the activation function is called an activation map. The activation map obtained by each kernel of a convolutional layer are stacked together and passed to the next layer. Layer after layer, the features gain in semantic level. Pooling operations can be introduced between layers to reduce the dimensions of the activation maps and at the same time avoid an overfit of the exact location of a feature. It consists in separately dividing each activation map into a grid and keeping for each cell its maximum or average value.

2.3.2 Residual neural network

Residual neural network (ResNet) [13] is a CNN widely used as a feature extractor, that has the particularity to use skip connections, which consists in adding the output of the previous convolutional layer to the next one, just before the activation function. The idea behind this comes from the observation that both the training and testing error of a network with 56 layers are higher than with only 20 layers.

We might assume that the deeper model could simply learn the same features as the lighter network in the first 20 layers and then learn an identity mapping so that its training error is not greater. As it failed to do so, reformulating the mapping to solve by adding the contribution of the last layer makes the identity mapping much easier to approach for the solver, as it now corresponds to weights of zero.

2.3.3 Receptive field

The receptive field (RF) of a layer in a CNN is the size of the region in the input image that has an influence on the neuron output. Thus, the RF is the same for all neurons in a given layer. The RF increases after each convolutional layer added to the network, and depends on the size of the kernels used to perform the convolution. This concept is illustrated in Figure 2.1.

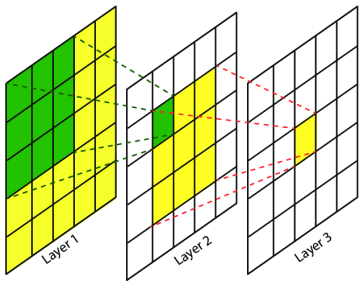


Figure 2.1: Illustration of the receptive field in a CNN [14]

Small objects can be perceived in early layers already, whereas larger objects need a bigger receptive field and often more complex features. However, to properly detect small objects, context information about its surroundings is also needed.

In the case of traffic lights, early layers of the feature extractor backbone are able to determine the state (color) and location of a traffic light, but might mistake a rear car light for a traffic light due to the small receptive field resulting in a lack of information about its wider environment [15].

One thing to note as well is the difference between the theoretical receptive field described briefly above and the effective receptive field (ERF) which measures the importance that each of the pixels within the receptive field have on the neuron output. It has been shown in [16] that it follows a Gaussian law in most of the cases, where the pixels at the center of the receptive field have more impact than the outer ones. Besides, in contrast to the theoretical RF, the ERF decreases after each convolutional layer, as it gives more and more importance to the central pixels.

2.3.4 Pyramid Scene Parsing Network

Pyramid Scene Parsing Network (PSPNet) [3] was introduced in 2017 and aim to increase the effective receptive field and add context to the final feature maps of a feature extractor backbone such as ResNet. The heart of the PSPNet is the Pyramid Pooling Module (PPM), illustrated in Figure 2.2. For different scales, adaptive average pooling is applied on these feature maps, followed by an up-sampling step to bring them back to their original

dimensions. Then, the newly obtained maps are concatenated to the original ones. This provides for each pixel multiple level of context: from global context (whole image) to local context (small sub-region around the pixel). In other words, by looking at all the feature maps at a same specific location, information about features concerning the pixels outside its receptive field is also provided.

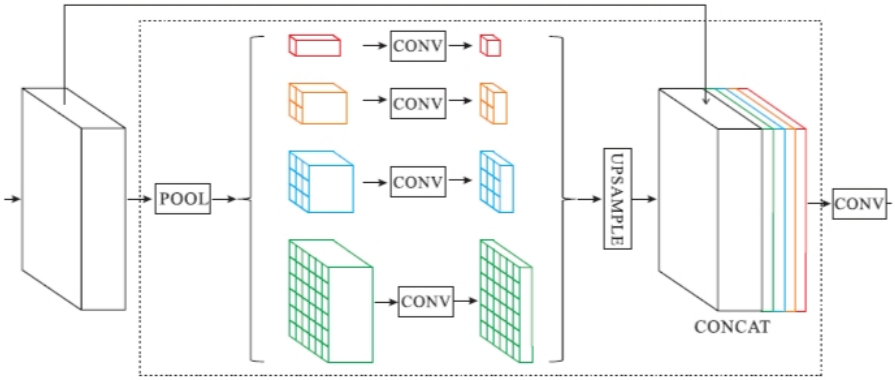


Figure 2.2: Architecture of a Pyramid Pooling Module [3]. Average pooling is performed on each of the backbone feature maps at different scales: from global pooling (in red), to a small sub-region pooling (in green).

2.3.5 Feature Pyramid Network

Feature Pyramid Network (FPN) [4], also introduced in 2017, is quite similar to PSPNet. However, its outputted feature maps are coming from different layers, not only the last one. They are obtained by concatenating the intermediate feature maps of one layer with the up-sampled feature maps of the next layer. Thus, combining the good spatial resolution of the earliest layers with the higher semantic level in the features of the latest. This enables to include more complex features in spatially more accurate maps, as well as considering small objects that would not have been detected with the feature maps of the latest layer only. The architecture is presented in Figure 2.3.

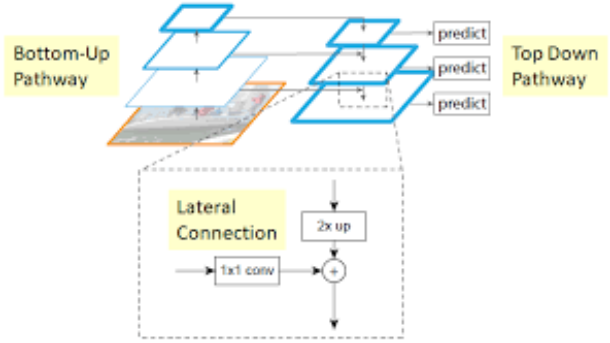


Figure 2.3: Architecture of a Feature Pyramid Network [4]

Chapter 3

Related work

The main concepts used for this thesis having been presented, we propose now an overview of the latest state-of-the-art and related previous work aiming to train in a end-to-end fashion an autonomous agent to drive using CARLA simulator.

3.1 Conditional imitation learning

3.1.1 Learning by Cheating

Learning by Cheating (LBC) [2] proposes a novel way of training an agent to learn a robust driving policy. It separates the learning process in two: learning to act, and learning to perceive using a teacher-student network approach.

1. A privileged agent (teacher) that has access to a ground truth 192x192x7 bird's-eye view (BEV) semantically segmented image (and thus does not need to learn perception but only to act) is trained offline with the supervision of a dataset of expert trajectories.

2. A sensorimotor agent (student) that has only access to a standard forward-facing 384x160x3 RGB picture is trained offline and online, with the supervison of the teacher. The learned student policy thus does not need any privileged information and is end-to-end.

Both the teacher and student agents have multiple prediction heads connected to the backbone, which outputs for each of the possible directional command (turn left, turn right, go straight, follow lane) heatmaps that are converted into waypoints using a soft-argmax operation. These waypoints are then converted to vehicle commands by a PID controller.

The dataset of expert trajectories is collected directly in the simulator using an agent based on CARLA autopilot. It collects both the BEV semantically segmented image and the forward facing RGB image. Data augmentation is performed by rotating the BEV to simulate steering noise. Multiple driving episodes are gathred in different traffic and weather conditions.

Having the teacher in between the expert and the student enables to train it for all possible commands at once and to have a supervision for any state of the environment, and not only the one shown by the expert. The teacher uses a randomly initialized ResNet18 backbone while the student uses a ResNet34 pre-trained on ImageNet.

When the paper was published in the end of 2019, it reduced the number of traffic lights infractions and collisions by an order of magnitude, compared to the state-of-the art on the CARLA simulator at the time, *Exploring the limitations of behavior cloning for autonomous driving* [5].

3.1.2 Cheating by Segmentation

Although the student of LBC could be transferred to the real-world without too much difficulties according to the authors, the teacher used to train it relies on the ground-truth BEV semantic segmentation image which could be difficult and expensive to gather. *Cheating by Segmentation* (CBS) [1] addresses this by replacing it by a 120deg field-of-view forward-facing camera. This would enable to use pre-existing large datasets such as Waymo [17] to train the student.

CBS gives promising results, but has difficulty anticipating braking actions, which results in a much higher amount of collisions and traffic light infractions than LBC on the *NoCrash* [5] benchmark. According to its author, it could be a consequence of the change of perspective in the segmentation camera from BEV to forward facing: close objects appear bigger which results in a shorter-term behaviour.

3.2 Reinforcement learning

3.2.1 World on rails

Learning to drive from a world on rails (WOR), published in 2021 by the same authors as LBC, proposes a model-based reinforcement learning method. It relies on a model of the ego-vehicle, which enables to simulate the outcome of the agent's actions. This model is learned by training a network to predict the next agent's state (location, orientation, speed) given its initial state and an action (steer, throttle, brake). This is done using pre-recorded driving logs from CARLA.

The main assumption of the paper is that the world is on rails, meaning that the agent has no influence on its environment. Consequently, the latter does not depend on the agent's state or actions which means the initial world state determines the entire sequence of world states. As the agent is unable to change this sequence, the state transitions for the world are simply the ordered sequence of pre-recorded world states from the driving logs.

The model of the ego-vehicle (and the known sequence of world states) are used alongside a reward function to compute the Q-value, $Q_t(s,\ a)$, for all possible combination of agent state s and action a at timestep t. The Q-value is the reward received for taking a given action in the current state plus the discounted estimated optimal return on the long term that it will get in the state it ends up in. It is called discounted, because it is weighted by a factor $\in [0, 1]$ to ponder it depending on how much it should care about future rewards. It is computed for each directional command.

The reward function is designed in a way to encourage the vehicle to stay within the target lane as well as to stop for a pedestrian or a traffic light, and is penalized otherwise. It once again relies on the information provided by the driving logs.

Finally, the Q-value table is used to supervise a visuomotor agent that takes a RGB image and the vehicle speed as input. The RGB image is fed to a feature extractor, flattened, concatenated with the speed and fed to fully connected layers. It is supervised on all directional commands.

The learned policy performs better than LBC, which was in the meantime established as the state-of-the-art on the *NoCrash* benchmark. Besides, it is able to learn to perform safe actions directly, whereas in imitation learning it is needed to show mistakes and recoveries.

In addition, it is more data efficient than model-free reinforcement learning approaches as it does not require to actually perform all actions thanks to the ego-vehicle model and the Q-value table.

Chapter 4

Approach

The approach taken in this thesis uses as starting point CBS: a teacher learns to drive from a dataset of forward-facing semantically segmented images and predicts waypoints in the image frame. It then supervises a student, that has the same image perspective as the teacher but with an RGB image input. Finally, the learned student model is used alone in the environment and its predicted waypoints are converted by a controller into vehicle commands.

In this thesis, the goal is to modify the student network architecture using two pyramid perception modules, PPM and FPN, previously presented in respectively Section 2.3.4 and 2.3.5. Besides, we bring the number of categories provided by the semantically segmented images from 13 to 5, keeping only the most important ones, which will be described in this chapter.

4.1 Environment

This work uses CARLA simulator, which provides a realistic driving environment and proposes various town models as well as multiple weather conditions. Besides, a wide range of sensors is available to integrate into the agent's vehicle. It is also possible to set the traffic conditions, such as the number of cars or pedestrians.

4.2 Data

4.2.1 Collection

The dataset consists in multiple driving episodes gathered by deploying a collector agent in the simulator. The agent is based on the one implemented in WOR and is given a set of waypoints to reach in a given CARLA town. Rewards are attributed to the agent for attaining these waypoints and for braking at red traffic lights as well as for pedestrians and cars.

Information about the agent's surroundings is based on the semantically segmented BEV retrieved from the simulator. For example, attributing a non-zero reward for braking for a pedestrian is possible only if the latter is visible in the BEV. The BEV is centered on the ego-vehicle and has a range of 32 meters in each cardinal direction. Finally, the agent

performs the action that maximize the Q-value, in order to optimize its return on the long term.

The dataset collected for this work is created by gathering at each timestep the following information:

- Ego-vehicle absolute world pose: `pos`: {x,y,z} [coordinates], `rot`: {pitch, yaw, roll} [deg]
- Ego-vehicle directional command: `cmd` $\in$ {1:Left, 2:Right, 3:Straight, 4:LaneFollow}
- Ego-vehicle forward speed: `spd` [m/s]
- Camera absolute world pose: `cam_pos`: {x,y,z} [coordinates], `cam_rot`: {pitch, yaw, roll} [deg]
- Semantically segmented image: `SSI` (384x160x5, 120deg fov)
- RGB image: `RGB` (384x160x3, 120deg fov)
- Bird's eye-view map: `BEV` (96x96x12)
- Relevant traffic light state: `RTLS` $\in$ {0:Green, 1:Yellow/Red}

The BEV is used only to compute the rewards for the collector agent. The semantically segmented and RGB images are used to train respectively the teacher and the student. The 5 channels of the semantically segmented image are the following (in parenthesis is given the corresponding CARLA label).

1. *Pedestrian* (#4)
2. *RoadLine* (#6)
3. *Road* (#7)
4. *Vehicles* (#10)
5. *TrafficLight* (#18) (post-processed)

Also, the *TrafficLight* channel of `SSI` does not encompass the state of the traffic light (in contrast to the BEV used for training the teacher in LBC that has a separate channel for each state). Therefore, CARLA built-in utility function is used to detect the relevant red traffic light state `RTLS` from a chosen threshold distance (3 meters). It is post-processed following CBS's approach, presented in (4.1).

$$SSI[i, j, \textit{TrafficLight}] = \begin{cases} \texttt{SSI}[\text{i, j, } \textit{TrafficLight}], & \text{if } \texttt{RTLS} = 1 \\ 0, & \texttt{otherwise} \end{cases} \tag{4.1}$$

Thus, the resulting channel shows the location of all the lights in the vicinity of the vehicle if the relevant traffic light is red. Otherwise, the channel is empty. The filtered *TrafficLight* channel of `SSI` according to the traffic light state is illustrated in Figure 4.1. This is a simplification made, since to have a proper red state channel, the locations of the non-red traffic light should be filtered out. As this does not prevent the teacher from learning correctly to stop, we keep this approximation.

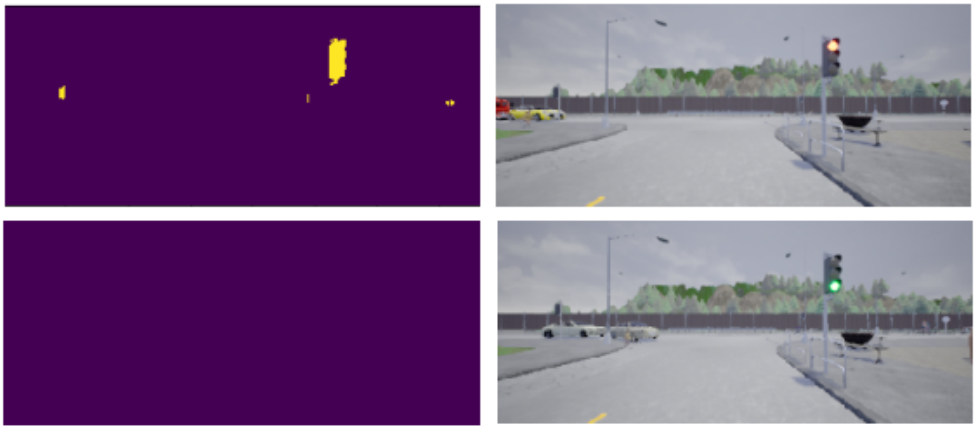


Figure 4.1: Left: post-processed *TrafficLight* channel of `SSI` according to (4.1). Right: corresponding `RGB` image. As only one channel is used for traffic light information in the semantically segmented image, their presence indicates to the vehicle that it must stop. Therefore, the traffic light is masked when it turns green.

Besides, the following changes are made to the WOR collector agent in order to be compatible with the CBS framework:

- Changing the camera position and resolution
- Adapting the order of the returned orientation angles from {roll, pitch, yaw} to {pitch, yaw, roll}
- The collector agent starts to brake too soon for a traffic light compared to what is needed for our implementation. Indeed, a non-zero reward for braking is attributed as soon as it appears in `BEV`. Thus, we filter out the `BEV` channel corresponding to the red traffic light using `RTLS` in the exact same way as done for `SSI` in (4.1). It is then given as input for the Q-value computation. Therefore, a non-zero reward for braking is now only possible if closer than 3 meters from the traffic light.

4.2.2 Ground truth waypoints

To compute the ground truth waypoints, which are used to supervise the teacher, the approach taken in CBS is employed, which consists in sampling the ego-vehicle pose at different timesteps:

Parameters:

- N_{STEPS}: Number of future waypoints to predict
- GAP: Gap between sampled timesteps

The recorded ego-vehicle trajectory is used to compute the ground truth waypoints: at timestep t, N_{STEPS} waypoints are obtained by sampling the future position of the vehicle every GAP timesteps. If the relevant traffic light is red and within the defined threshold (`RTLS` $= 1$), all the waypoints are placed on the ground just in front of the vehicle and thus lead to a stop. This is done by adding to the vehicle position an offset in the direction

of its forward vector. This offset is found heuristically and depends on the height of the camera.

Doing this enables to make it clear that the vehicle should stop when the light is red. As the expert driver starts stopping when `RTLS` = 1, there are a few frames where it is still moving. Also, the waypoints would otherwise anticipate a restart just before the light turns green, which would mislead the network as there are no visual cues that the light is about to change its state. The formula for the waypoints computation is presented in (4.2).

$$w_{t,k} = \begin{cases} \texttt{pos}[t] + 5.5 \cdot [cos(\texttt{rot.yaw}), sin(\texttt{rot.yaw}), 0], & \text{if } \texttt{RTLS} = 1 \\ \texttt{pos}[t + k \cdot GAP], & \texttt{otherwise} \end{cases} \tag{4.2}$$

where $k \in [0,\ N_{STEPS}\ \text{-}\ 1]$

The waypoints are then converted to the image coordinate system with the help of the camera pose. It can happen that some of them are not visible in the forward-facing camera perspective, for instance just before a turn. In this case, the missing ones are found by interpolation directly in the image frame. But if there are less than two waypoints available and an interpolation is not possible, they are replaced by waypoints leading to a stop (as for a red traffic light). Figure 4.2 shows the RGB image captured by the ego-vehicle's forward camera, with the ground truth waypoints overprinted, that is used for debugging purposes. Figure 4.3 shows the situation where the ground truth waypoints are not computed from the recorded vehicle trajectory but imposed to lead to a stop.

Figure 4.2: Ego-vehicle's driver perspective RGB image used for debugging with ground truth waypoints overprinted in blue. In yellow, from left to right: the relevant traffic light state (`RTLS`), the forward speed (`spd`), the directional command (`cmd`), as well as the method used to generate the waypoints are also displayed. In this case, it is 'InFrame', which means that all waypoints are already visible in the camera frame and there is no need for interpolation.

Figure 4.3: As the traffic light is red, the waypoints are imposed to lead to a stop even though the vehicle is still moving (`spd=3.54`). This allows to ease the task of the teacher network to associate red traffic lights with a stop.

4.2.3 Augmentation

Data augmentation follows the approach of WOR. It is applied during data collection by:

- Adding Ornstein-Uhlenbeck (O.-U.) noise to the ego-vehicle steering wheel angle
- The weather conditions are constantly changing during the episode

The following augmentations are performed afterwards on the RGB image, each with a probability of 20% and in a random order:

- Gaussian blur
- Additive Gaussian noise
- Pixel dropout
- Scaling
- Linear contrast
- Conversion to grayscale
- Elastic transformation (pixels moved around locally)

As we use imitation learning and not reinforcement learning, the agent cannot explore at will and learn from it, but is restricted to the experiences of the expert. Therefore, the latter has to show how to act correctly in an unusual situation. Adding noise to the expert steering command demonstrates how to recover and get back to the desired trajectory. However, adding to much noise can make the agent drive in zigzag, as shown in Figure 4.4. For this reason, the noise intensity is reduced compared to what is done in WOR. The value is found heuristically by decreasing it until the shaky behaviour disappears. Ornstein-Uhlenbeck process is Gaussian, therefore it can be defined by its amplitude θ, mean μ and standard deviation σ.

Figure 4.4: Ornstein-Uhlenbeck noise set with the initial WOR values $\{\mu = 0, \sigma = 0.1, \theta = 0.1\}$. The trained agent learned to imitate the steering noise of the expert, and repeatedly goes into the opposite lane. The noise must be reduced to avoid this behaviour but not too much to still be able to learn to recover.

4.3 Network Architecture

This section presents the network architecture on which this thesis relies. The learning process uses a teacher-student approach. The teacher is supervised by the ground truth waypoints presented before and, once trained, is used to supervise a student. Figure 4.5 illustrates the whole framework. In the following sub-sections, the teacher and student are presented in details.

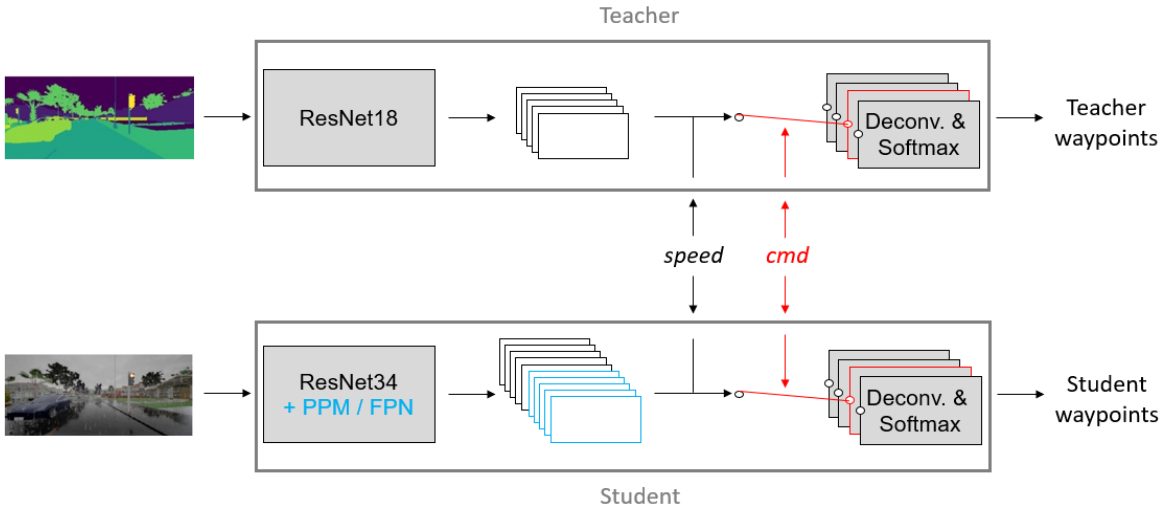


Figure 4.5: Illustration of the network architecture. The teacher takes a semantically segmented image as input, while the student gets an RGB image. Feature maps are obtained by passing it through a feature extractor. They are concatenated with the speed and fed to one of the four waypoint prediction heads according to the directional command. The teacher is supervised by the ground truth waypoints and the student by the teacher's. If a pyramid module (PPM or FPN) is included, there are twice as much output feature maps.

4.3.1 Teacher

The teacher architecture is unchanged from CBS. The network gets as input the forward speed of the ego-vehicle `spd` and the forward-facing semantically segmented image `SSI` representing roads, road lines, red traffic lights, cars and pedestrians. The image is fed into a Resnet18 backbone and its output at the 4th layer, just before the fully connected layers, is retrieved. It is then concatenated with a repeated version of the forward speed `spd` and fed to one of the specialized waypoint prediction head, according to the directional command `cmd`.

These prediction heads perform deconvolution to predict heatmaps that are converted into waypoints in the image frame using a spatial arg-softmax. The model is supervised by the ground truth waypoints coming from the dataset. The loss function is the mean squared error (MSE) between the network predictions and the ground truth waypoints, measured in a rescaled image space of [-1,1] x [-1,1]. Using the MSE instead of the L1 distance used previously helped successfully stopping at traffic lights. In such a situation, the ground truth waypoints are just at the bottom of the image frame (as explained in Section 4.2.2). Therefore, wrongfully predicting to follow the route results in a very high loss. This change from L1 distance to MSE is mainly for the student: when trained on the dataset described previously, it used to barely recognize any red traffic light. If the teacher already managed to predict to stop for traffic lights with the L1 distance, having the same loss function is more convenient to evaluate the learning process.

4.3.2 Student

The student uses the same framework as the teacher, except that it takes an RGB image as input and uses a ResNet34 backbone instead of a ResNet18. It is supervised by the waypoints predictions of the teacher. As mentioned before, the MSE between the two sets of waypoints is used for the loss function, instead of the L1 distance as in CBS.

The main objective of this thesis is to improve the traffic light perception in the network, without handling them separately, to keep the method end-to-end. Therefore, two different additional modules are tested and integrated between the existing backbone and waypoint prediction heads of the network. These modules have their implementation detailed hereafter.

Pyramid Pooling Module

The PPM is the heart of the PSPNet introduced in Section 2.3.4 and performs adaptive average pooling of feature maps at different scales, before up-sampling them and adding them to the existing ones.

This module is integrated into the architecture by placing it after the ResNet34 backbone but before the concatenation with the ego-vehicle speed. It thus takes as input the feature maps of the last ResNet34 convolutional layer. Figure 4.6 details its implementation, which corresponds to the illustration shown in Section 2.3.4. The bins chosen for the adaptive average pooling are the one proposed by the authors [3].

Using PPM should provide a more robust representation of the image by fusing together features from different sub-regions. Its authors have established its effectiveness against mismatched relationships. In our case, a potential mismatched relationship would be to

confound a car rear light for a red traffic light. If they look quite alike, they are disposed in a very different context. They also demonstrated its ability to distinguish superposed objects that have the same color. This could help noticing earlier an obstacle that blends into the backgrounnd.

PPM
In: Last convolutional layer feature maps (ResNet34, layer 4) [512, I·$\frac{1}{32}$, I·$\frac{1}{32}$]

For each of the N bins:
AdaptiveAvgPool2d(bin): [512, I·$\frac{1}{32}$, I·$\frac{1}{32}$] → [512, bin, bin]
Conv2d: [512, bin, bin] → [512/N, bin, bin]
BatchNorm2d
ReLU
Interpolation: [512/N, bin, bin] → [512/N, I·$\frac{1}{32}$, I·$\frac{1}{32}$]
Concatenation

Out: [1024, I·$\frac{1}{32}$, I·$\frac{1}{32}$]

Where `I` represents the image dimensions.

Parameters
Bins: [1x1, 2x2, 3x3, 6x6]

Figure 4.6: Pyramid Pooling Module implementation and parameters

Feature Pyramid Network

The FPN, introduced in Section 2.3.5, takes as input the feature maps of each layer of the backbone, in our case ResNet34. Thus, it gets feature maps with different numbers of channels and dimensions. The first step is to convolve each of these inputs with *feature_maps_out* = 256 kernels of size (*feature_maps_in* x 1 x 1) in order to get for each backbone layer the same amount of channels, while keeping their original dimensions.

Then, a top-down pathway, starting from the last layer enables to merge these layer outputs together, using lateral connections. The last layer feature maps are up-sampled by a factor 2, making them the same size as the ones of the previous layer and allowing them to be merged together by element-wise addition. The result of the addition is convoluted with a kernel of size 256x3x3 (and padding of 1), to counter the aliasing effect caused by the up-sampling [18]. The result is then up-sampled again and merged with the layer before, and so on until the first layer is reached. It results in a set of feature maps of the same dimensions as the backbone intermediate layers outputs, but with richer features thanks to the multi-scale merging.

To be able to integrate it to the existing student network without needing to change it, we propose to add a down-sampling step so that each set of feature maps has the same

dimensions as the last backbone convolutional layer. Finally, we concatenate them to end up with a unique output of the same dimensions as PPM. The resulting architecture is described in Figure 4.7 and illustrated in Figure 4.8.

FPN

In: Feature map from all ResNet34 backbone layers:

C_i: [c_i, dim_i, dim_i]
C1: [64, I·$\frac{1}{4}$, I·$\frac{1}{4}$]
C2: [128, I·$\frac{1}{8}$, I·$\frac{1}{8}$]
C3: [256, I·$\frac{1}{16}$, I·$\frac{1}{16}$]
C4: [512, I·$\frac{1}{32}$, I·$\frac{1}{32}$]

P_4 = Conv2d(C_4): [c_4, dim_4, dim_4] → [256, dim_4, dim_4], k=(1, 1), s=(1, 1)

For i ∈ {3, 2, 1}:

A = Interpolation(P_{i+1}): [256, dim_i, dim_i] → [256, 2·dim_i, 2·dim_i]
B = Conv2d(C_i): [c_i, dim_i, dim_i] → [256, dim_i, dim_i], k=(1, 1), s=(1, 1)
P_i = Conv2d(A $\bigoplus$ B): [256, dim_i, dim_i] → [256, dim_i, dim_i], k=(3, 3), s=(1, 1), p=(1,1)

For each of the merged feature maps P_i:

Down-sampling: [256, dim_i, dim_i] → [256, $dim_{i=4}$, $dim_{i=4}$]

Concatenation

Out: [1024, I·$\frac{1}{32}$, I·$\frac{1}{32}$]

Where `I` represents the image dimensions, `k` the kernel size, `s` the stride, `p` the padding and $\bigoplus$ the element-wise addition.

Figure 4.7: Feature Pyramid Network implementation

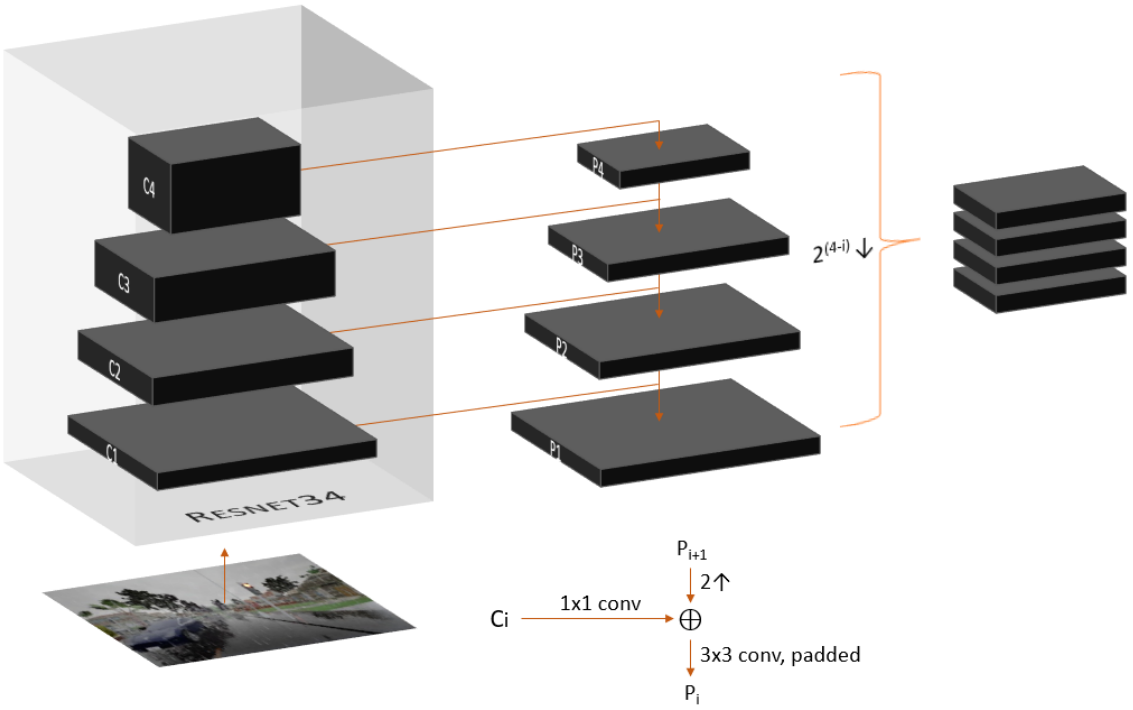


Figure 4.8: Illustration of the proposed Feature Pyramid Network architecture. An RGB image is fed to the ResNet34 backbone. Feature maps are extracted at every layer. They go through a 1x1 convolution to have all the same number of channels. They are iteratively merged together by an element-wise addition with the upsampled merged feature map of the next layer. Obtained merged feature maps are then downsampled to match the dimension of the ResNet34 last layer. This enables to keep the same architecture for the student afterwards.

By fusing together high semantic - low resolution features of the latest layers with low semantic - high resolution of the earliest layers, a more accurate representation of the image is constructed, which should help the waypoint prediction heads in their task.

4.4 Controller

The controller responsible for transforming waypoints to vehicle commands is the same as the one used for LBC and CBS. The target velocity is the average speed needed to go from one waypoint to the next one. Given the current speed of the vehicle, a longitudinal PID controller must then compute the throttle and brake commands that shall make it reach its target speed. The steer command is computed by fitting an arc through the predicted waypoints. The angle formed by the current position and the projection of one of the waypoints on the arc is fed to a lateral PID controller. The index of the chosen waypoint for the projection depends on the directional command: the closer the waypoint, the smaller the turning radius will be.

Chapter 5

Experimental setup

The approach having been described, we present now the setup and dataset used to train our networks, as well as the evaluation process used to measure the performances of the different models. We compare two different modules that can be added to the existing student network architecture, in order to favour a better recognition of traffic lights. The process remains end-to-end and no classification is made. Both modules can be introduced between the backbone and the waypoint prediction heads. The code used for the project is briefly presented in Appendix B.

5.1 Setup

We first used the version 0.9.6 of CARLA which is employed in LBC and CBS. However, in order to have reproducible runs, we switched in the middle of the project to 0.9.10.1. Indeed, this version enables to fix the random seed responsible for traffic generation (other cars and traffic light state), which is helpful for analysing the results. It also introduces an evaluator that is now used by the latest state-of-the-art implementations, allowing us to compare performances with them on common metrics.

We decided to still include some interesting results obtained with CARLA 0.9.6. The approach is the same, except that the models are trained on a different dataset and with the L1 distance instead of the MSE for the loss function. Besides, the controller responsible for converting waypoints into vehicle commands has slightly different parameters. Only PPM is evaluated as FPN required a newer version of *torch* to extract the feature maps at intermediate levels of the backbone, incompatible with CARLA 0.9.6. Finally, it performs data aggregation, as in LBC. As the ablation study made in CBS did not measure an improvement in the generalization capabilities, we decided not to port this feature to our CARLA 0.9.10.1 implementation. The results for CARLA 0.9.6 are available in Appendix A.1. The following of this thesis is dedicated only to the CARLA 0.9.10.1 implementation.

5.2 Dataset

The dataset is collected using the agent detailed in Section 4.2.1 and follows the approach of WOR, but has been collected especially for this work. It is then split into two distinct datasets used respectively for training and validation, with a 80%-20% ratio. Table 5.1

summarizes its characteristics and Table shows 5.2 the values of the ground truth waypoints parameters.

Dataset 0.9.10.1	
Collector strategy	WOR (Q-Value)
Location	*Town01*
Weather	#1, #3, #6, #8
Vehicles	120
Pedestrians	150
Number of frames	70k train, 15k val
Steering noise	O.-U. $\{\mu = 0, \sigma = 0.015, \theta = 0.015\}$
Steering noise probability	100% of frames
Target speed (straight)	6.5m/s
Target speed (curves)	6m/s
Traffic light detection	from 3m

Table 5.1: Summary of the dataset characteristics and parameters

Ground truth waypoints parameters	
N_{STEPS}	5
GAP	5

Table 5.2: Parameters used to generate ground truth waypoints

In addition, Figure 5.1 presents some additional insights about the dataset's contents. As we can see, frames representing different situations are quite equally distributed between training and validation set.

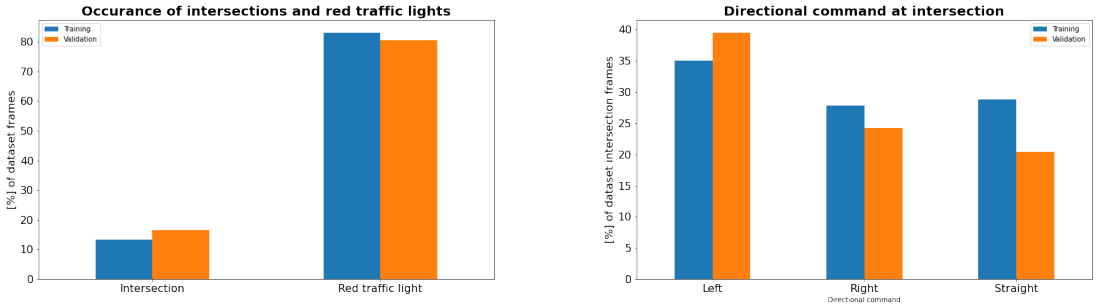


Figure 5.1: Analysis of the frames in the training (blue) and validation (orange) datasets. Left: percentage of frames showing intersections and red traffic lights. Right: distribution of the frames within the 3 possible directional commands at intersections.

5.3 Training

The teacher is trained on the dataset described previously and supervises three different students:

- *Original*: without additional module
- *PPM*: with the Pyramid Pooling Module proposed in 4.6
- *FPN*: with the Feature Pyramid Network proposed in 4.7

The parameters used to train the teacher and students are described in Table 5.3.

Teacher		**Students**	
Backbone	ResNet18 (random init.)	Backbone	ResNet34 (ImageNet init.)
Batch size	64	Batch size	96
Optimizer	Adam	Optimizer	Adam
Learning rate	1e-3	Learning rate	1e-4
Weight decay	0	Weight decay	0

Table 5.3: Network parameters used to train the teacher and the student

5.4 Controller

Compared to LBC and CBS, the braking threshold is incremented from 2.0m/s to 3.8m/s in order to adapt the controller to the sensitivity our model has to obstacles. Its value is tuned in the train town with train weather conditions, using the student without additional module. It is a trade-off between stopping for more obstacles and risking to block the agent for a false obstacle such as a puddle of water on the road and get stuck as it will not eventually move like a car would. Chapter 7 further develops the subject. Moreover, the target speed is clipped to be in a range from 0m/s to 5m/s as in CBS.

5.5 Evaluation

Our models are evaluated on the *NoCrash* benchmark. As suggested by its name, a run is stopped as soon as the agent collides. It is also the case if it deviates by more than 30 meters from the route or if it does not move for 180 seconds (timeout). *NoCrash* assesses the performances of the agent in train (*Town01*) and test (*Town02*) environments, as well as train (#1, #3, #6, #8) and test (#10, #14) weather conditions. For all of these combinations, the agent is always evaluated in three different traffic conditions: *Empty*, *Regular* and *Dense*, detailed in Table 5.4

Configuration	**Vehicles**	**Pedestrians**
Town01-Empty	0	0
Town01-Regular	20	50
Town01-Dense	100	250
Town02-Empty	0	0
Town02-Regular	15	50
Town02-Dense	70	150

Table 5.4: Number of vehicles and pedestrians under the three traffic conditions, *Empty*, *Regular* and *Dense*, depending on the town.

There are 25 predefined routes in each town that are evaluated in every traffic and weather conditions. Each of these runs is called an episode. For instance, in the test town with test weather conditions, there are 25 routes x 4 weathers x 3 traffics resulting in 300 episodes. Therefore, a complete benchmark consists in total in 900 episodes. Figure 5.2 illustrates an example of route from both towns.

Figure 5.2: Example of route from *Town01* (left) and *Town02* (right) [6]

We compare the performances under the following metrics with WOR, the leading implementation on *NoCrash* and the former state-of-the-art, LBC.

- *Route completion*: % of the itinerary accomplished, averaged over all episodes
- *Success Rate*: % of episodes where the goal is reached before the time limit, which is defined as the time required to attain the destination at a speed of 5km/h
- Number of traffic lights infractions per hour

Moreover, we provide these additional metrics:

- *No Collision Rate*: % of episodes ended without collision
- *No Block Rate*: % of episodes ended without the vehicle being stuck (timeout of 180s)
- *In Lane Rate*: % of the itinerary where the vehicle stayed in its lane, averaged over all episodes

For all the metrics, the higher is the better, except for the number of traffic lights ran per hour.

Chapter 6

Results

The results of the experiments previously introduced are presented hereafter. The learning process is first described and followed by the evaluation of the selected models on the *NoCrash* benchmark.

6.1 Learning curves

6.1.1 Teacher

Figure 6.1 shows the training and validation curves of the teacher. The 50th epoch checkpoint (loss of respectively 0.0028 and 0.015) is chosen to supervise the student network. Training it for longer lead to an increased validation loss that signify an overfit of the training dataset.

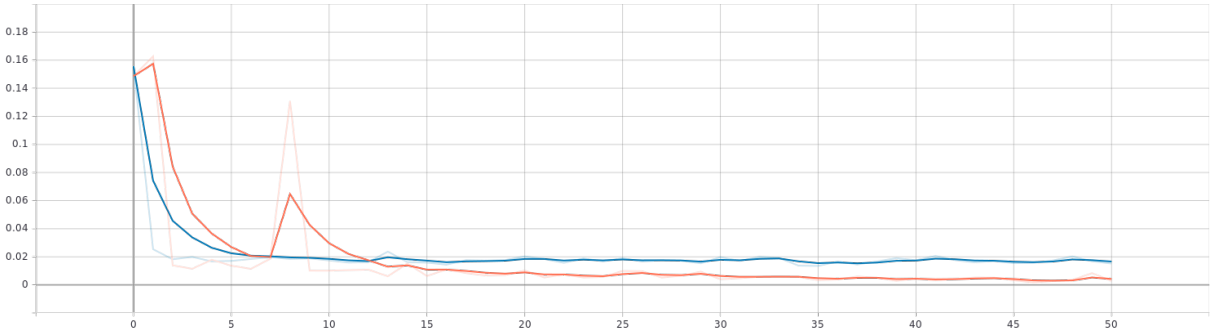


Figure 6.1: Training (orange) and validation (blue) curves of the teacher

Figure 6.2 presents examples of predicted waypoints at the latest log before the epoch chosen as final model. They are overprinted on the semantically segmented image that is fed to the network as input. The two other inputs, which are the current speed of the ego-vehicle and the directional command are also displayed.

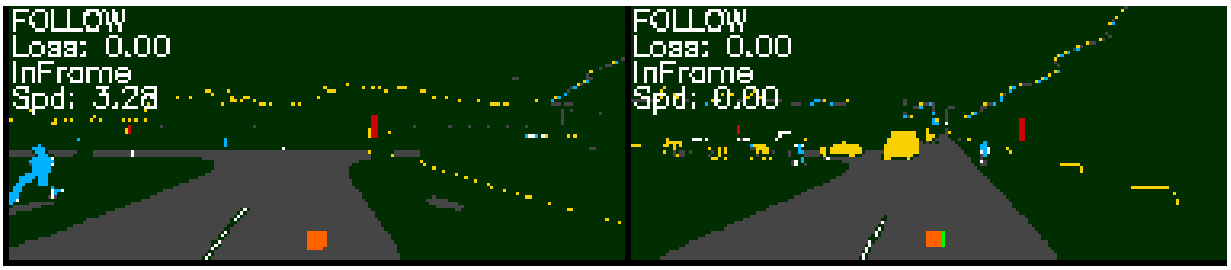


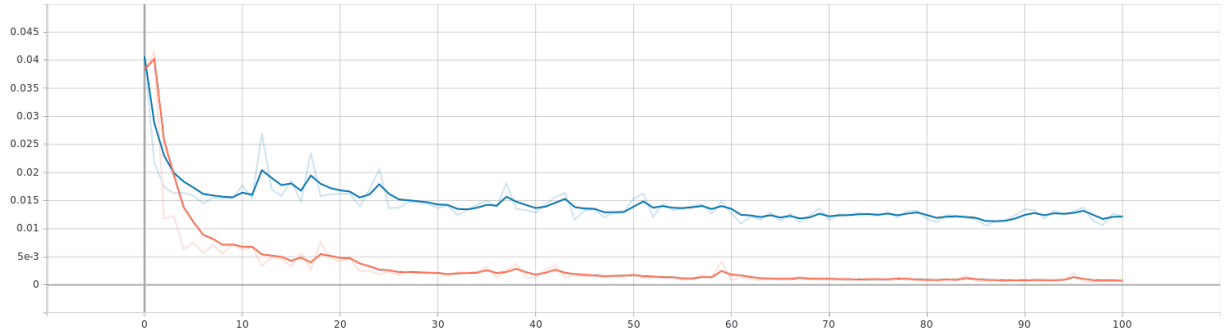
Figure 6.2: Semantically segmented image with ground truth (green) and predicted (orange) waypoints. Here, they are superposed because the teacher correctly predicted to stop for the red traffic light (in red).

6.1.2 Student

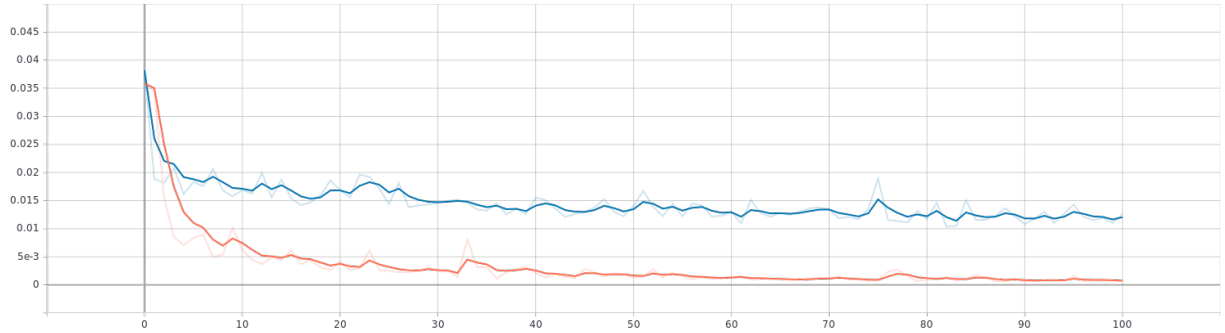
Three different student models are trained with the supervision of the teacher obtained in the previous section. One without additional module, one with a PPM, and one relying on a FPN.

The learning curves are presented in Figure 6.3. The chosen model is the one with the lowest validation loss after maximum 100 epochs. They are presented in Table 6.1.

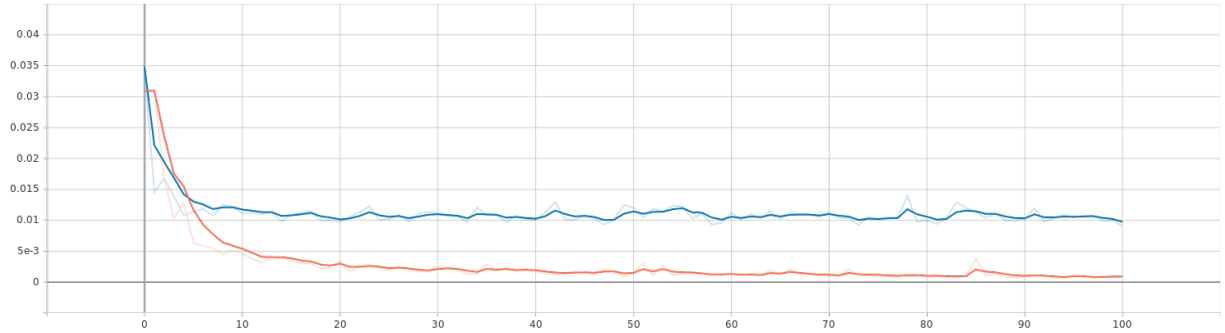
All models have a similar validation loss, which, although it is not increasing, stays quite constant and maintain a non-negligeable gap with the training curve. This could indicate a small overfit of the dataset. To overcome this in the future, the regularization could be strengthen for instance by performing more data augmentation or by introducing a weight decay in the optimizer.



(a) Original



(b) PPM



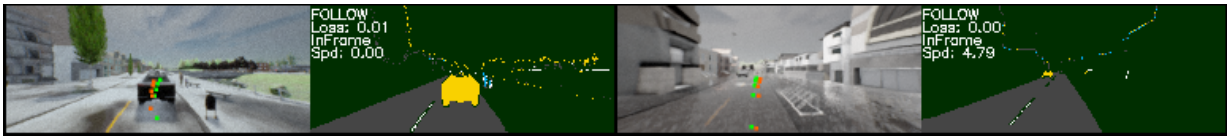
(c) FPN

Figure 6.3: Training (orange) and validation (blue) curves of the three students

Model	Epoch	Train	Val
CBS2	86	0.0007	0.0106
$CBS2_{PPM}$	90	0.0006	0.0108
$CBS2_{FPN}$	100	0.0009	0.0090

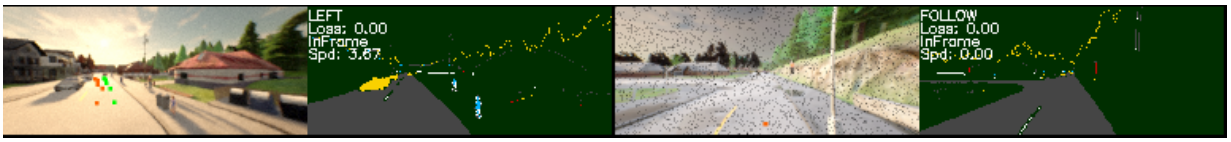
Table 6.1: Loss of the chosen student models. It is computed as the MSE between the teacher and student predicted waypoints measured in a rescaled image space of size [-1, 1] x [-1, 1].

Figure 6.4 presents examples of predictions for the three different student models. They are compared to the predictions of the teacher used to supervise them. We can also observe the results of the data augmentation performed on the RGB image fed to the student.



(a) Original

(b) PPM



(c) FPN

Figure 6.4: Semantically segmented and RGB image given as input to respectively the teacher and the student during validation. Predicted waypoints are overprinted on the RGB image (green: teacher, orange: student).

6.2 *NoCrash* results

We evaluate the three student models selected in the previous section on the *NoCrash* benchmark and compare them with the results of WOR and LBC as published in [6]. The *Route Completion*, which measures the percentage of the itinerary completed is presented in Table 6.2. The *Success Rate*, which counts the percentage of episodes with a *Route Completion* of 100% and finished before the time limit, is presented in Appendix A.2 for reference.

	Train town						Test town					
	Train weather			Test weather			Train weather			Test weather		
	Empty	Reg	Dense	Empty	Reg	Dense	Empty	Reg	Dense	Empty	Reg	Dense
LBC	97	96	91	79	79	76	92	88	75	62	64	45
WoR	**99**	**100**	**98**	**95**	**93**	**96**	**99**	**95**	**89**	**85**	**89**	**81**
CBS2	**46**	47	31	34	39	28	32	39	30	28	31	32
$CBS2_{PPM}$	24	24	20	21	23	20	16	20	19	14	1	17
$CBS2_{FPN}$	45	**53**	**43**	**40**	**45**	**35**	**42**	**50**	**38**	**37**	**48**	**39**

Table 6.2: Mean route completion on NoCrash [%]

The *Route Completion* results are well below the state-of-the-art. This is not due to the vehicle leaving the road or even its lane. Indeed, the performances on lane keeping are excellent in every configuration, as shown in Figure 6.5.

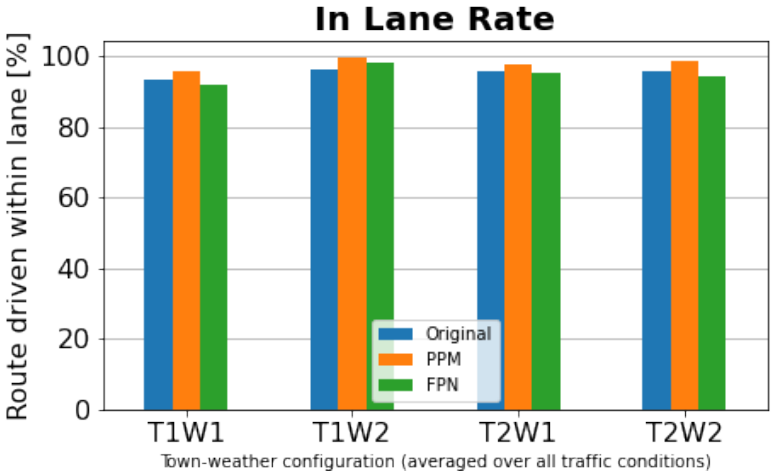


Figure 6.5: Percentage of the route driven within the lane, according to the town-weather configuration (1:train, 2:test). All agents are very reliable and stay on track in every environment.

The main reason for this quite low route completion is that the sensitivity to other vehicles is moderate. This means that the network reduces its target speed but often not sufficiently to avoid a collision. That is why the braking threshold had to be increased as mentioned in Section 5.4. Therefore, we choose to collide less, but at the same time risk getting stuck due to a "false obstacle" such as a puddle of water. This trade-off is discussed further in Chapter 7. Besides, Figure 6.6 presents the percentage of episodes ended without being blocked.

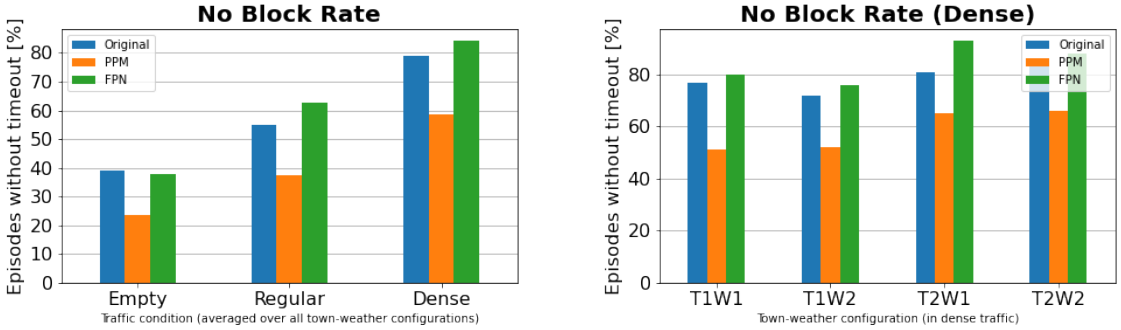


Figure 6.6: Percentage of episodes ended without being blocked. This is defined to be the case when the vehicle does not move for 180 seconds. Left: The vehicle is less stuck in heavy traffic as it generates a change in the environment seen by the agent, creating a chance to make it move. Right: If we consider only the *Dense* traffic, we notice that it is less likely to be blocked in the test town.

We can observe that the agent is less likely to be blocked in heavy traffic, as the resulting change in the visible environment is the only reason for it to make a change in its waypoint predictions, thereby giving it a chance to move. Moreover, it is clear that the PPM model is more often stuck. This could be explained by the fact that it is more capable than the others to differentiate objects from the road even if they have a similar texture. These are referred to as inconspicuous objects in [3]. This results in the vehicle getting stuck more often for irregularities on the road (such as crevices or water). Nonetheless, PPM is

better at perceiving cars. This is illustrated in Figure 6.7, where the percentage of episodes ended without collision is depicted. We have to focus on the *Dense* scenario to observe differences in the three models performances, as they equally hardly ever leave the road and thus rarely collide with non-traffic obstacles.

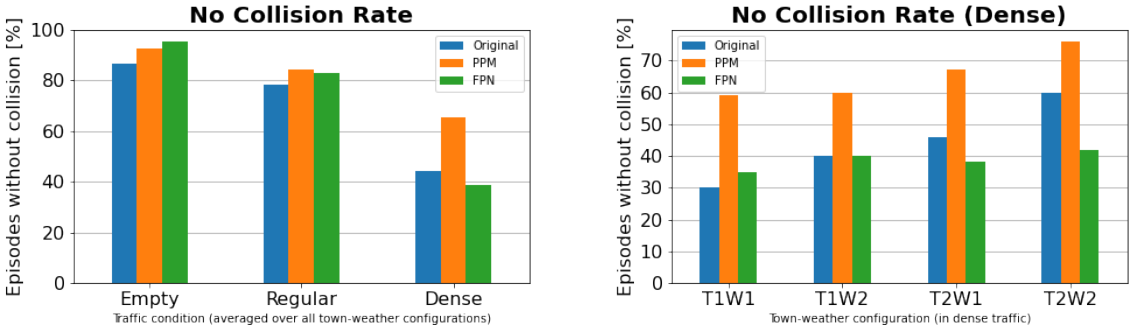


Figure 6.7: Left: Percentage of episodes ended without collision according to the traffic condition. Right: Same but only in *Dense* traffic, according to the town-weather configuration. PPM performs better than the two other models. It risks of course less collisions than the others since it is more often stuck, as seen in Figure 6.6. But this difference is not big enough to explain alone the better results of PPM at avoiding collisions, which are also due to its better perception of obstacles.

To conclude the results presentation, we take a look at the traffic light infractions which are detailed in Table 6.3 and summarized in Figure 6.8.

	Train town						Test town					
	Train weather			Test weather			Train weather			Test weather		
	Empty	Reg	Dense	Empty	Reg	Dense	Empty	Reg	Dense	Empty	Reg	Dense
LBC	1.35	1.89	3.27	0.36	0.81	**0.52**	**8.45**	8.22	**7.26**	**8.17**	**8.61**	**4.87**
WOR	**0.00**	**0.43**	**2.61**	**0.00**	**0.00**	4.29	10.68	**6.95**	12.90	14.46	11.30	13.28
CBS2	15.07	18.45	20.7	16.74	18.06	16.1	19.27	20.1	24.34	25.23	24.94	16.28
$CBS2_{PPM}$	3.37	5.71	4.37	14.9	16.66	20.9	17.84	13.87	10.12	**2.53***	24.62	18.05
$CBS2_{FPN}$	**1.15**	**2.12**	**2.51**	**0.49**	**1.65**	**1.57**	**6.70**	**9.13**	**8.69**	7.74	**10.70**	**13.27**

Table 6.3: Number of traffic lights infractions per hour

* Note: the infraction score of the PPM agent in the *Empty* scenario of the test town and weather is underestimated due to it being significantly more stuck (72% of the episodes) compared to the other traffic scenarios.

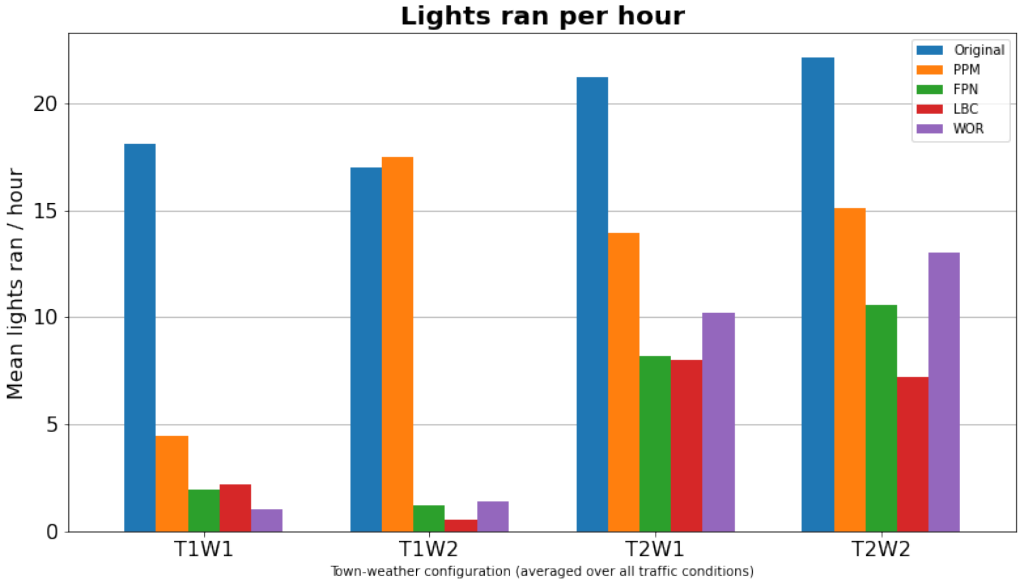


Figure 6.8: Number of traffic light infractions per hour of driving according to the town and weather configuration. PPM brings improvement to the original CBS implementation and FPN enables to have an infraction rate comparable to WOR and LBC.

FPN provides a robust traffic light handling, competitive to WOR and close to LBC. Fusing feature maps from different layers of the backbone enables it to recognize more precisely where and which features in the image are responsible for stopping or restarting. Also, it generalizes well to unseen environments, with a number of infractions per hour of only 10.57 in the test town - test weather configuration. Besides, it seems that the weather conditions have a very limited impact on its performances.

Note that the dataset and loss function used to train the network play an important role too. In this case, changing the loss function from L1 distance to MSE had a very positive impact on the traffic light handling for the three agents.

To sum up, the collected dataset and the training method leads to models that all demonstrates a very good lane keeping. Using PPM and especially FPN allows to significantly reduces the number of traffic light infractions. However, if all models see their target speed decrease when approaching another vehicle, it is not sensitive enough yet. The next chapter presents observations on the behaviour of the different agents and proposes ideas for improvement.

Chapter 7

Discussion

Presented hereafter are observations on the behaviour of the vehicle regarding obstacle detection and traffic light handling as well as possible solutions to issues often encounterd.

7.1 Sensitivity

As our approach is end-to-end, the reactions to different kind of obstacles or even different instances of it are not binary. In a modular fashion, if we simplify a little bit, there are only two possible outcomes. Either a car is detected at a certain position and the agent stops at a fixed distance to it, or it is missed and the agent crashes into it.

However, our only outputs are waypoints that defines the target speed. When the target speed is reduced, the kind of obstacle responsible for this is unknown. Therefore, it is not straightforward to define a unique speed threshold under which the vehicle should stop.

In our case, the agent responds more strongly to cars than pedestrians, therefore a trade-off has to be found. Of course, the braking threshold can be heighten so that even slightly detected obstacles are stopped for. But it should not be too high, otherwise the vehicle will not be able to restart once the obstacle gone. Figure 7.1 shows the FPN agent braking for pedestrians.

(a) Target speed starts decreasing as pedestrian detected

(b) Stop completed

(c) Restarts but quite early, a lower braking threshold could have been dangerous

Figure 7.1: FPN agent stopping for pedestrians

A possibility to increase the sensitivity to both vehicles and pedestrians would be to pursue the approach that CBS has with the traffic lights: imposing the ground truth waypoints to lead to a stop when a car or pedestrian is within a distance below a threshold, to emphasize the need for braking.

7.2 False obstacles

One of the main remaining issue is due to the agent braking for non dangerous obstacles. For instance, in some weather conditions, when water accumulates to form puddles, as illustrated in Figure 7.2.

Figure 7.2: A puddle is formed on the ground in one of the test weather condition. Due to the change of appearance on the road, our model interprets it as an obstacle and reduces its target speed.

We notice that all models suffer from this. Therefore, further developing the RGB image data augmentation, by adding spots of more considerable size to simulate these kind of irregularities, could be beneficial.

Besides, it should be easier for the vehicle to restart when it is stuck for a non-traffic related reason. Currently, the expert rarely shows how to restart from a random spot in the street, but just how to do it if there is an actual reason for stopping, like a car or a traffic light. Therefore, the model suffers from an issue: if given an image of an empty road with a high current speed it will predict waypoints leading to a high target speed as well. But if the current speed is zero, the target speed is often below the braking threshold, thus the vehicle is blocked. Adding sequences with the expert going from zero to full speed in random places should help solving the issue.

Nonetheless, solving the issue of sensitivity to vehicles and pedestrians will consequently solve this as well, as it will be possible to place a low enough braking threshold that would not make the vehicle stop for these false obstacles anymore.

7.3 Influence of other vehicles

This paragraph presents some observations related to the impact other vehicles have on the predicted waypoints. Two specific situations make our agent plan a higher target speed. These phenomenons are observed when the agent is stopped.

Firstly, having the vehicle in front at a safe distance (but still visible in the image) make our agent predict a higher target speed than if there were no vehicle at all. For instance, if it is waiting in a queue, the vehicle in front restarting will make it increases its target speed. This phenomenon is desired and can be explained by the fact that the expert used for the data collection keeps a fixed minimal distance with the vehicle in front and accelerates towards its cruise speed otherwise.

A similar phenomenon happens when our vehicle is stopped outside an intersection and a vehicle drives past on the opposite lane. During its passage, the target speed of our

vehicle temporarily increases, as shown in Figure 7.3.

(a) Agent stuck on the road as the target speed is below the braking threshold

(b) Vehicle in the opposite lane makes the target speed increase

(c) Agent starts moving

Figure 7.3: Vehicle in the opposite lane results in higher target speed

These situations have no known undesired effects, it can even help restart a stuck vehicle. However, it is important to know that these phenomenons exist to understand the behaviour of the agent.

7.4 Traffic lights

The traffic lights are now well recognized, especially in PPM and FPN. But even in the best of the three, FPN, there remains a handful of occurrences where it is missed. Figure 7.4 and 7.5 respectively illustrates a successful traffic light handling and an infraction.

(a) Target speed starts decreasing as TL detected

(b) Stop completed

(c) Restarts in the first frame where the traffic light is green

Figure 7.4: Successful stop

(a) Traffic light detected but later

(b) Low speed, but not completely stopped

(c) Traffic light now too close to be noticed as expert always stops before that

Figure 7.5: Infraction

The main lesson learned from this example of infraction is that, despite the model starts detecting the traffic light late in (a), it would have been able to stop in time, had it not ceased perceiving the traffic light in (c). To solve this issue, we could imagine having an expert that stops at different distances from a red traffic light, going from the current threshold to just below the pole. This would allow to learn a more robust policy as it gives a longer range of distance where braking is the expected behaviour. This would be useful in case the model does not perceive the traffic light directly, for instance due to an unknown weather condition. Besides, relying on a method such as VisualBackProp [19] to visualize which input pixels had the most impact on the predicted output could give additional insight on how to improve the agent.

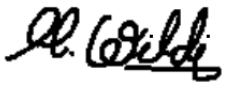
Chapter 8

Conclusion

In this thesis, we trained an agent end-to-end to imitate an expert using only images taken from a driver perspective. Two different student network architectures were tested. Both the Pyramid Pooling Module and the Feature Pyramid Network resulted in an agent presenting a very good lane keeping. PPM enabled to react more strongly to cars but also to inoffensive obstacles. Making the ground truth waypoints lead to a stop in the vicinity of cars and pedestrians would enable the teacher to provide a more robust supervision which would benefit to all of the student architectures tested. This could also allow to decrease the braking threshold and lead to a more fluid agent.

FPN enabled to significantly reduce the number of traffic lights ran, reaching a number of infractions per hour of 10.6% in test conditions on *NoCrash*. This brings it close to the state-of-the-art implementations *Learning to drive from a world on rails* and *Learning by Cheating*. Diversifying the distance at which the expert stops for a traffic light could further improve the model.

This thesis has shown insights on the potential of pyramid perception modules for conditional imitation learning and hopefully their perception abilities will contribute to the improvement of autonomous driving.



Maël Wildi

Bibliography

[1] T. van Orden, "Cheating by segmentation." Bachelor thesis, Universiteit van Amsterdam, June 2021.

[2] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, "Learning by cheating," *Conference on Robot Learning (CoRL)*, 2019.

[3] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[4] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[5] F. Codevilla, E. Santana, A. M. López, and A. Gaidon, "Exploring the limitations of behavior cloning for autonomous driving," *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019.

[6] D. Chen, V. Koltun, and P. Krähenbühl, "Learning to drive from a world on rails," *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[7] A. Tampuu, T. Matiisen, M. Semikin, D. Fishman, and N. Muhammad, "A survey of end-to-end driving: Architectures and training methods," *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, 2020.

[8] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 2016.

[9] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," *Proceedings of the 1st Annual Conference on Robot Learning (PMLR)*, 2017.

[10] S. Chowdhuri, T. Pankaj, and K. Zipser, "Multinet: Multi-modal multi-task learning for autonomous driving," *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2019.

[11] Y. Chen, J. Wang, J. Li, C. Lu, Z. Luo, H. Xue, and C. Wang, "Lidar-video driving dataset: Learning driving policies effectively," *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[12] A. Prakash, K. Chitta, and A. Geiger, "Multi-modal fusion transformer for end-to-end autonomous driving," *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[14] H. Lin, Z. Shi, and Z. Zou, "Maritime semantic labeling of optical remote sensing images with multi-scale fully convolutional network," *Remote Sensing*, 2017.

[15] J. Müller and K. Dietmayer, "Detecting traffic lights by single shot detection," *21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018.

[16] W. Luo, Y. Li, R. Urtasun, and R. Zemel, "Understanding the effective receptive field in deep convolutional neural networks," *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS)*, 2016.

[17] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset," pp. 2446–2454, 2020.

[18] S.-H. Tsang, "Review: Fpn - feature pyramid network (object detection)," *Medium*, March 2019.

[19] M. Bojarski, A. Choromanska, K. Choromanski, B. Firner, L. J. Ackel, U. Muller, P. Yeres, and K. Zieba, "Visualbackprop: Efficient visualization of cnns for autonomous driving," *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[20] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

Appendix

A Additional results

A.1 CARLA 0.9.6

This section presents the experiment carried out in CARLA 0.9.6 where a PPM is added to the student network architecture, just after the feature extractor, as was later done in CARLA 0.9.10.1. The aim is to add context to these extracted features, in order to improve the perception of traffic lights. Its performances on the *NoCrash* benchmark are compared to those of the original model, which does not use any additional module.

Dataset

The dataset used in 0.9.6 is the exact same as CBS and follows the approach of LBC, where the collector agent is based on CARLA's autopilot.

Dataset 0.9.6	
Collector strategy	LBC (based on CARLA)
Location	*Town01*
Weather	#1, #3, #6, #8
Vehicles	100
Pedestrians	250
Number of frames	167k train, 39k val
Steering noise	U$\sim (-0.2, 0.2)$
Steering noise probability	10% of frames
Target speed (straight)	6m/s
Target speed (curves)	4.5m/s
Traffic light detection	from 5.5m

Table 1: Summary of the dataset characteristics and parameters

Evaluation

- % of episodes ended without collision
- Traffic light success rate: % of traffic lights correctly handled by the vehicle (green lights included)

- Red traffic light success rate: % of red traffic lights correctly handled by the vehicle (green lights excluded)

NoCrash Results

As shown in Figure 1 (left), the percentage of traffic lights correctly handled in the *NoCrash* benchmark increases when using PPM, especially in the test town. This phenomenon is even clearer if we exclude the green traffic lights and focus only on the percentage of red traffic lights successfully handled, as shown in Figure 1 (right). In that case, PPM enables to double the amount of correctly handled lights. This shows that using multi-scale features allows to create a model able to better generalize to a new environment.

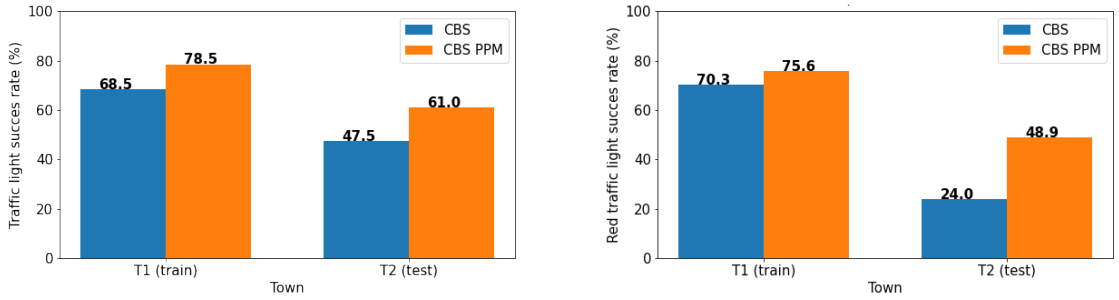


Figure 1: Percentage of traffic lights handled without infraction (left: including green lights, right: excluding green lights) in the train and test town of the *NoCrash* benchmark. For each town, 3 different weathers and 2 different random seeds are used.

Note that there is a counter-intuitive small gap (PPM: from 78.5 to 75.6) or even improvement (Original: from 68.5 to 70.3) of the traffic light success rate in the train town when excluding the green traffic lights (which should obviously never yield to an infraction). This is because for a few traffic light locations in the train town (*Town01*), the ground truth fails to tell when the relevant traffic light is red, and they are therefore considered as green but still counted as infraction if violated. In this experiment, it concerns between 5-10% of the cases (depending on the configuration, as the traffic light state is not reproducible between different runs in CARLA 0.9.6).

Figure 2 presents an example where PPM managed to stop in contrast to the original architecture that missed the traffic light.

Figure 2: *NoCrash* benchmark, *Town01*, *Regular* traffic, weather #14, seed 0. Left: PPM. Right: no PPM. Top: frame when relevant traffic light is red according to `RTLS`. Bottom: 15 frames after. In this situation, PPM enabled to stop for the traffic light.

Figure 3 shows the number of timesteps elapsed between the ground truth detection of a red traffic light and the agent's reaction to it (when there is one). Intuitively, this depends mostly on the waypoints used to supervise the model. Indeed, they set a higher and lower bound for the distance from a red traffic light at which to stop. Hence, the network has to learn a model able to associate these waypoints leading to a stop to features from an RGB image captured at a certain range from the corresponding red traffic light.

We can see that the results are quite similar for the two approaches, although PPM enables to react on average one timestep earlier. One of the main change is the variance in the number of timesteps needed to react in the train town: it is done with PPM in 50% of the cases within a range of 3 timesteps against 6 in the original implementation. This might confirm that using multi-scale feature maps enables to add context and thus provides more general features. This has the advantage of making the agent more predictable.

Note that we cannot say much about the variance of timesteps needed in the train town as, after more than 8 timesteps, there is no successful handling of the traffic lights since the model is not taught to stop this close from a traffic light.

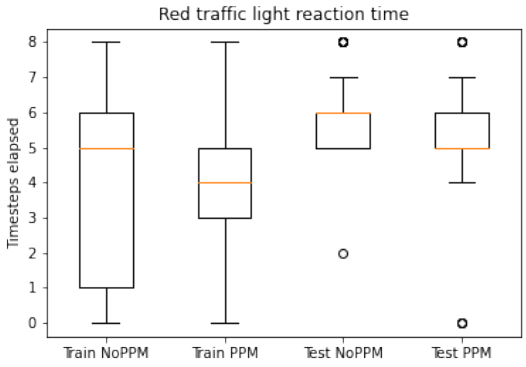


Figure 3: Timesteps elapsed between the appearance of a red traffic light (according to ground truth) and the vehicle's reaction to it. We consider this to be when the vehicle has cut its default target speed by a factor 2, which results in a speed of 2.5 m/s.

If the goal of this experiment is to improve the detection of small objects that are traffic lights, it is also critical to assure that it is not in the detriment of bigger objects such as other cars and that the model is indeed able to perceive objects of different scales.

Therefore, Figure 4 presents the number of episodes ended without any collision with obstacles such as cars or pedestrians under multiple traffic conditions. The drop in performances from train to test town is very narrow for PPM, meaning it favours a better generalization of obstacles under all traffic conditions. In addition, the good result in the test town with *Empty* traffic (52% with PPM, 27% without) testifies that PPM permits a reduction of collisions with non-cars obstacles, that is collisions due to the vehicle leaving the road.

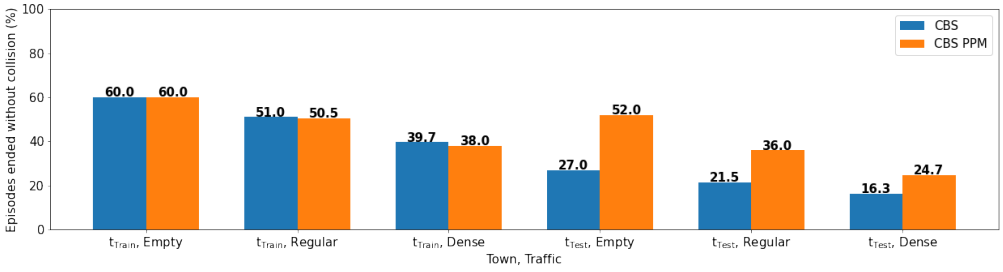


Figure 4: Percentage of episodes ended without any collision under different traffic conditions

To summarize, PPM mainly helps perceiving traffic lights that would otherwise not be detected at all (Figure 1, right) but decreases only very slightly the amount of timestep needed to react to it (Figure 3). Furthermore, this improvement in the perception of small objects like traffic lights is done without diminishing the perception of bigger elements such as cars, pedestrians or sidewalks. In fact, it even permits to improve the amount of episodes ended without collisions in unseen environments (Figure 4).

A.2 CARLA 0.9.10.1

Table 2 presents the *Success Rate* on *NoCrash* measured in the evaluation made in Chapter 6.

	Train town						Test town					
	Train weather			Test weather			Train weather			Test weather		
	Empty	Reg	Dense	Empty	Reg	Dense	Empty	Reg	Dense	Empty	Reg	Dense
LBC	89	87	75	60	60	54	86	79	53	36	36	12
WoR	**98**	**100**	**96**	**90**	**90**	**84**	**94**	**89**	**74**	**78**	**82**	**66**
CBS2	**18**	18	3	**12**	14	4	7	10	1	**6**	**10**	**8**
$CBS2_{PPM}$	2	1	0	0	0	0	2	3	1	0	0	0
$CBS2_{FPN}$	16	**27**	**10**	8	**16**	**6**	**17**	**19**	**7**	4	8	2

Table 2: *Success Rate* on NoCrash [%]

B Code

This section presents the code used for this project, available on GitHub [1]. It is based on Learning by Cheating (LBC) and Cheating by Segmentation (CBS). It follows the work of CBS which replaced the bird's-eye view semantically segmented image used to train the teacher in LBC.

B.1 Code source

This repository contains code that has been adapted from other sources :

- Evaluation:
  - CARLA Leaderboard
  - CARLA Scenario Runner

  (Instructions on how to setup CARLA and the leaderboard / scenario runner available below and at https://leaderboard.carla.org/get_started)
- Data collection, training, visualisations:
  - World on rails
  - Cheating by Segmentation (branch: segmentation)
  - Learning By Cheating

[1] https://github.com/mael25/CBS2/tree/cbs2

B.2 Installing CARLA

Install CARLA in the desired location:

```
wget https://carla-releases.s3.eu-west-3.amazonaws.com/Linux/CARLA_0.9.10.1.tar.gz
tar -xvzf CARLA_0.9.10.1.tar.gz -C carla09101
```

Clone CBS2 repository (branch: cbs2) in desired location:

```
git clone git@github.com:mael25/CBS2.git
cd CBS2
git checkout cbs2
```

If not already done, install conda and then create and activate this new virtual environment:
conda env create -f docs/cbs2.yml conda activate cbs2
To add more packages to the environment:

- **conda**: `conda install <package>`
- **conda-forge**: `conda install -c conda-forge <package>`
- **pip**: `~/anaconda3/envs/cbs2/bin/pip install <package>`

Add the following environmnet variables to `~/.bashrc`:
Verify the setup by launching CARLA (with cbs2 virtual environment activated):

```
source ~/.bashrc
$CBS2_ROOT/scripts/launch_carla.sh 1 2000
```

B.3 Launching CARLA

Make sure to start CARLA in another terminal before proceeding to data collection or evaluation.

```
$CBS2_ROOT/scripts/launch_carla.sh <num_runners> <port>
```

Note: if there are multiple runners, `<port>` is also the increment between them.

B.4 Data collection

Data collection configuration (settable in `autoagents/collector_agents/config_data_collection.`
- **num_per_flush**: Amount of timestep data per save - **noise_collect**: Add noise to the steering command of the agent - **main_data_dir**: Location where the data should be saved
- ...

To start the data collection:

```
python -m rails.data_phase1
```

B.5 Training

Teacher

```
cd cbs0/training
python -m train_birdview --segmentation --dynamic
 ↪  --batch_size=<batch_size>  --dataset_dir=<path_to_data_dir>
 ↪  --log_dir=<path_to_log_dir> --max_epoch=<max_epoch>
```

Student

Phase 0: warm-up stage

```
cd cbs0/training
python -m train_image_phase0 --log_dir=<path_to_log_dir>
 ↪  --teacher_path=<path_to_teacher_dir/model-XX.th>
 ↪  --dataset_dir=<path_to_data_dir>
```

Phase 1: actual training

```
cd cbs0/training
python -m train_image_phase1 --log_dir=<path_to_log_dir>
 ↪  --teacher_path=<path_to_teacher_dir/model-XX.th>
 ↪  --ckpt=<path_to_phase0_student_dir/model-XX.th>
 ↪  --dataset_dir=<path_to_data_dir> --pretrained --max_epoch=<max_epoch>
```

To train a model with PPM add for example `--ppm=1-2-3-6` (where [1,2,3,6] are the desired bin sizes used for the adaptive pooling). For FPN, add `--fpn`. These arguments must be added both for phase 0 and phase 1.

B.6 Evaluation

- To evaluate on CARLA Leaderboard:

```
python -m evaluate
```

To evaluate a model trained with PPM or FPN, add respectively `--mod=ppm` or `--mod=fpn`. This will change the configuration file used. The default configuration file for each type of model are located in the `results_lead` folder. The results of the evaluation are saved there as well.

- To evaluate on NoCrash:

```
python -m evaluate_nocrash --town=<TownXX> --weather <test/train> --resume
```

B.7 Project file structure

The tree structure of the important folders and files as well as their usage is reproduced in Figure 5.

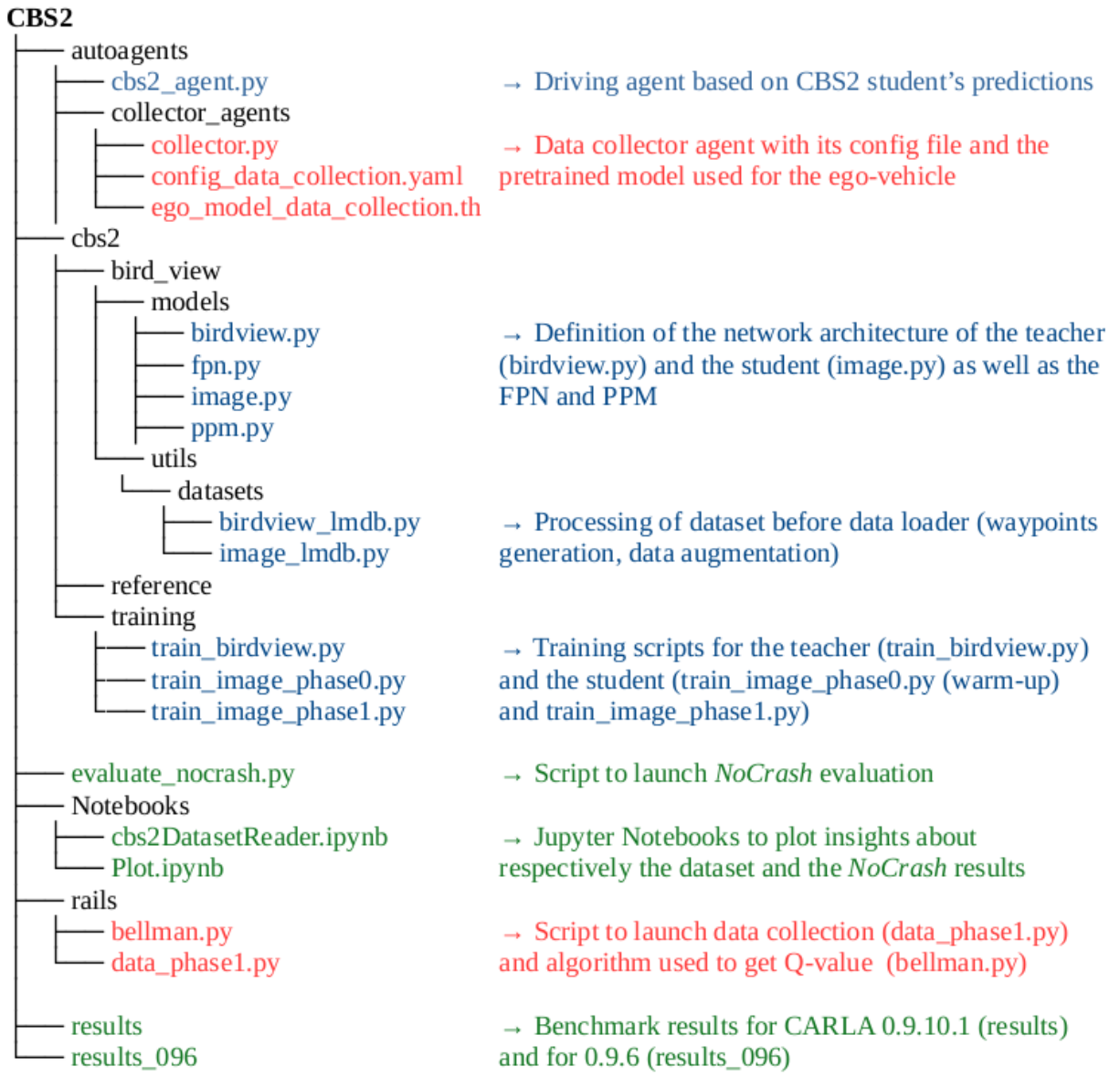


Figure 5: Tree structure of important folders and files for this project, separated by usage