End-to-End Imitation Learning From The Future

Eslam MOHAMED ALI SAID
islam.bakr.2017@gmail.com

1 Introduction

1.1 Project Overview

Deep learning is contributing greatly in many Automotive applications like: autonomous driving (AD), and augmented reality (AR). automatic emergency braking (AEB) [1], lane keeping assist (LKA) [2], active cruise control (ACC) [3], traffic jam assist (TJA) [4], and crash avoidance (CA). Forward collision warning (FCW) and automatic emergency braking (AEB) are considered as the initial trials to integrate crash avoidance [5] functionality. In FCW functionality, the vehicle has the ability to only warn the driver that there is an object in front of you, so the driver either takes the responsibility of taking reasonable actions. However in AEB, the vehicle starts taking action through braking in case of the vehicle comes near to front object. These actions are taken by the ego-vehicle based on integrating advanced sensors like: Laser, Camera and Radar, but these vehicle actions lead to occurrence of many crashes, in addition to being source of congestion because of its poorness.

Fully autonomous driving is one of the difficult problems faced the automotive applications. It is forbidden due to the presence of some restricted Laws that prevent cars from being autonomous for the fear of accidents occurrence. However, researchers try to reach autonomous driving as a new area for research for the aim of having a strong push against these restricted laws. Crash avoidance functionality is considered as one of the most important features in Self-Driving Cars. Recently, it is partially integrated in the self-driving car system.

Navigation is one of the most important problems which autonomous driving cars face, as it is a safety critical task, and needs full knowledge of the surrounding environment. In the last ten years, this topic was tackled by researchers, there are two approaches to tackle this problem, single stage and multi stage pipelines. The single stage approach is the End-To-End approach [6] [7] [8] [9], which tackle the autonomous driving problem as one block instead of split it to many blocks. This block is corresponding of generating the suitable controls directly from the sensors reading. The multi stage approach decomposes the problem into the following blocks: a) perception which is responsible for perceive the surrounding environment, b) trajectory prediction for the surrounding objects, c) trajectory planning which compute the trajectory depends on the perception and prediction for the environment, and d) control block which is responsible for taking the good actions depending on the whole information which are gathered by the previous block. The main disadvantages of this approach are that the error is accumulated through the pipeline, this means that small errors at earlier stages in the pipeline will be propagated through other stages.

1.2 Literature Review

Autonomous driving is considered as a hot field of research recently. [2] proposed autonomous driving framework based on using deep reinforcement learning solving either markov decision process (MDP) or partially observer markov decision process (POMDP). It depends on sensor fusion for the aim of having Robust readings for the surrounded environment. It controls the vehicle's manipulated parameters: throttle, steering angle by either continuous action algorithms or discrete action algorithms. [6] proposed an end-to-end behavioral cloning solution taking 3 front camera images installed on the ego-vehicle dashboard, then applied some sort of feature extraction via convolution layers, solving a regression problem controlling the vehicle's steering angle. [9] proposed conditional imitation learning by introducing heads idea for training a deep model on straight, left, right, and follow-lane . It decomposed each task of them into heads switching between them via control signal sent by Carla Simulator. However, [6] and [9] proposed solution are for used lane keeping assist functionality not for either autonomous driving or path planning. This is because of not interactions with dynamic objects in the environment like other vehicles, pedestrians, cyclists, etc.

Path planning and decision making for autonomous vehicles in urban environments enable self-driving cars to find the safest, most convenient, and most economically beneficial routes from a point to another. Finding routes is complicated by all of the static and dynamic maneuverable obstacles that a vehicle must identify and bypass. The major path planning approaches include the predictive control model, feasible model, and behavior-based model. A path is a continuous sequence of ahead way-points generated defining possible trajectory. A maneuver is a high-level characteristic of a vehicle's motion, encompassing the position and speed of the vehicle on the road, maneuvers include going straight, changing lanes, turning, and overtaking. Maneuver planning aims to take the best high-level decision for a vehicle while taking into account the path specified by path planning mechanisms. Trajectory planning or trajectory generation is the real-time planning of a vehicle's move from one feasible state to the next, satisfying the car's kinematic limits based on its dynamics and as constrained by the navigation mode.

Single stage approach is the end-to-end solutions that depends only on an input, and output the control actions (steering, throttle, and brake). In [10], Dean Pomerleau et al. depend only on a front camera and a laser range finder image to allow the vehicle following the lane. Huazhe Xu et al., in [11], depend on the motion model of the vehicle, GPS sensor, front camera, and the previous control action in order to output a reinforcement learning policy taking the next control action. In [12], Simon Hecker et al. depend on TOMTOM offline saved Map, GPS and a surrounding camera views to take the best control. This means that this paper merges between the traditional way of path planning with the deep learning solution, however this map needs to be updated frequently to avoid any change happened which is inefficient. We are proposing camera cocoon covering 360 degrees around the vehicle, and using different input representations for the captured images to achieve path planning and automatic emergency braking (AEB) functionalities through deep neural network model.

Multistage traditional approach as in [13] and [14] is composed of 4 cascaded blocks: 1) localization block, 2) sensor fusion block, 3) waypoints generator block, and 4) control block. Localization block localizes the vehicle based on a GPS sensor in a predefined map. Sensor fusion block achieves a robust localization for the vehicle, and better environment perception for the surrounded objects. Many sensors can be fused together as lidar, radar, and camera. Waypoints generator is responsible for generating waypoints ahead knowing the driving freespace, so that it could slow down when there is another vehicle in front of it or it could maneuver. Control block takes the reasonable vehicle's control actions by using PID controller, or model predictive control (MPC) controller. Recently published papers are following the previous multistage approach with different combinations of the defined blocks. Some use sensor fusion between only lidar and radar, others use lidar only, others use camera as in [15] and [16], and others integrate it with lidar. Some researchers use PID controller, others use MPC controller, others use hybrid controller between both of them.

Autonomous driving requires essential parameters needed to be satisfied in order to have a proper functionality. There must be a defined map for the environment, various sensor fusion like between GPS; camera; lidar; and radar, and way-points trajectory generations. The vehicle is localized depending on either urban area or highway environment defined map, and depending on GPS sensor knowing the current vehicle lane in the road. Environment perception is ensured using sensor fusion between different sources like camera, radar, and lidar achieving robust sensor readings. Camera aims to classify both dynamic and static objects in the road, while radars and lidars are fused to classify objects and know the distance away from other objects. The vehicle depends on the previous in order to generate way-points ahead in all of the possible routes given that knowing locations of other objects. Every point represent a state, so the vehicle moves from one state to another showing the progress in the vehicle motion. An efficient finite state machine is constructed to take proper actions: keep going in the same lane, increase the vehicle speed gradually, overtake the ahead object so change the lane, decrease the speed gradually, do not change the lane due to other object is coming from the adjacent Lanes. Model predictive control (MPC) or PID controller can be used in order to control vehicle's control actions like steering angle and throttle.

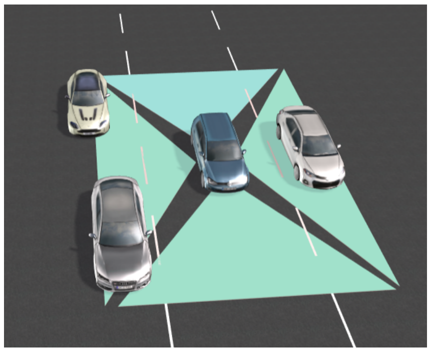
Figure 1: Autonomous driving vehicle

In this work, I will tackle self driving car problems following the End to End approach, by building a deep learning agent which will be able to control the vehicle by outputting throttle, steer and brake value. It was proven in several occurrence that taking historical information will be a huge added value to the agent robustness. Historical information may be the direct previous sensor readings or the distribution of the network outputs. There are a lot of ways to achieve that for example by using Recurrent neural networks or even by simply stacking the previous frames from the sensor readings. What if taking the advantage of the future too, using video prediction techniques, we could predict the future frames on the basis of previous frames. then stacking them with the current and the previous frames to increase our model robustness.

1.3 Metrics

An ablation study will be made to compare the effect of stacking previous frames and the predicted future frames with the current one. In this study we will make use of the collected data-set from [9], to train different agents to be able to compare among them to study the effect of stacking previous and future frames to determine the best combination percentage to achieve the highest accuracy.

To achieve that we need a well shaped metric to compare among different agents, so the benchmark which is mentioned in [9] which is built using CARLA simulator [17] is used to evaluate the performance of each agent. The evaluation process cover different aspects which enable us to judge on the agent well like:

- The total number of the succeed scenarios where the agent is spawned in deterministic location and it's goal to reach to the desired destination.
- The total travelled distance by the agent.
- Evaluation process covers different weathers and conditions which is never seen in the training by the agent.

2 Analysis

2.1 Data Exploration

Any deep learning model to be trained well, It is mandatory having a huge data-set for the training process, so in this section the data which is used in training will be illustrated in details. There are a lot of navigation data-sets which are open source, some of them are real data-sets and others are synthesis data.

2.1.1 Real Data

In this section i will go throw a real data-sets which could be used in autonomous driving projects and list it's advantages and disadvantages. KITTI [18], Cityscapes [19], Comma.ai, Berkeley DeepDrive and Baidu Apolloscapes datasets were collected using forward camera only so they missed the information around the vehicle as there are no more cameras around the vehicle (COCOON

CAMERA). Although there are another datasets which cover the surrounding view as Oxford's Robotic Car [20], Argoverse [21] and nuScenes [22] datasets, but those data don't contain vehicle's action labels like steering angle and throttle, but they provide us with IMU reading which could be used to generate vehicle's action labels like the steering and throttle. Drive360 [12] contains vehicle's action labels and the surrounding view around the vehicle. All these data covers highway road, urban and cities beside that they are collected using a calibrated sensors and the vehicle is driven by an expert driver.

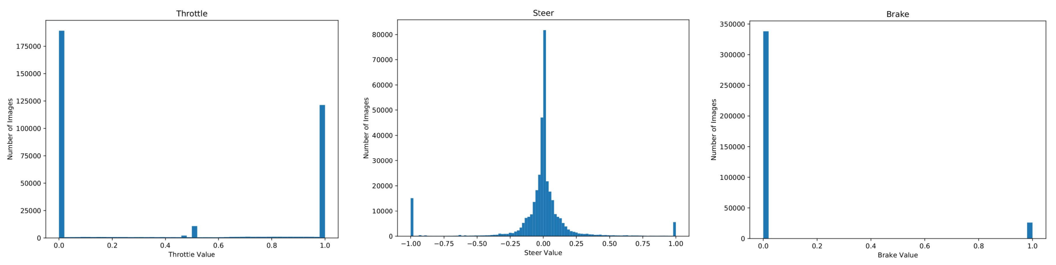

Figure 2: Training data distribution

2.1.2 Synthesis Data

However there are a lot of real data-sets but simulated data is heavily used in the autonomous driving field, specially nowadays there are a lot of car simulators like [17] and [23] which enables the developers to evaluate their algorithms in a rich environment or even create your own environment. Another reason which make the synthesis data is a power full option in this field is their is no need for expert driver to collect the data, as this process is very expensive but using car simulators their is no need for that as simulator's autopilot is used in data collection phase, beside that their are some field in deep learning like reinforcement learning which reacquires interaction between the agent and the environments while the training process is running and of course it is impossible to use this method in real world for safety aspects.

In this project I used this data which is used in [9] to train multi head neural network which contains four heads, training dataset comprises 2 hours of human driving in Town 1 of which only 10% (roughly 12 minutes) contain demonstrations with injected noise. Collecting training data with strong injected noise was quite exhausting for the human driver. However, a relatively small amount of such data proved very effective in stabilizing the learned policy. The four heads and the architecture are illustrated in 3.1 and also I used the same bench-marker which is used in [9] to compare my approach with their results.

2.1.3 Data Analysis

The collected data by [9] which is used in my project is collected using CARLA simulator [17]. Taking the advantage of CARLA's auto pilot they collected a huge data set in different weathers conditions and CARLA towns. After determine a random start and end position the auto pilot controls the vehicle to reach the desired destination while the navigation process a data is collected. The collected data contains sensor reading , which are RGB camera and lidar sensor and besides the sensor reading a lot of measurements are collected like steering angle in this time stamp, the throttle, the brake value, the town name, is collision is occurred or not, the percentage of crossing the road boarders, the weather condition, the forward speed of the vehicle and the high level command which represents the direction of the car, this high command could comes from the drivers or from another sensor like GPS sensor, the high level command controls switching the network heads as discussed in 3.1. Noise is injected into the steering during data collection. Namely, at random points in time we added a perturbation to the steering angle provided by the autopilot. The perturbation is a triangular impulse: it increases linearly, reaches a maximal value, and then linearly declines. This simulates gradual drift from the desired trajectory, similar to what might happen with a poorly trained controller.

The triangular impulse is parametrized by its starting time t_0, duration $\tau \in R^+$, $\sigma \in \{$-1,+1$\}$, and intensity $\gamma \in R^+$ as in the following equation:

$$S_{perturb}(t) = \sigma\gamma \max\left(0, \left(1 - \left|\frac{2(t - t_o)}{\tau} - 1\right|\right)\right) \tag{1}$$

then apply the three actions to the vehicle including the augmented steering angle, but saving the original steering angle before adding the noise to it. Then the data-set is splitted to training data set and validation data set as follows 80% and 20% respectively.

As shown in figure 2 the training data distribution as data analysis is an important phase, as we must make sure our data is correct before beginning the training process. As shown in figure 2 we could say that the data is well and we could use it for training as the left image in figure 2 shows us the throttle distribution, at the center the distribution of the steering angle which makes sense as the steering is centered around zero in the majority of the time and finally the last image on the right shows us the brake distribution.

3 Methodology

3.1 Network Architecture

Mainly in this project two architectures are used as follows:

- The first one is Stochastic Adversarial Video Prediction (SAVP) [24] which is responsible of taking the current camera frame and previous frames then generate predicted future frames.
- The second one inspired by 4 is the end to end self driving car architecture which is responsible about taking the stacked images and the vehicle speed as inputs and outputs the vehicle's direct actions which in our case is the throttle, steering angle and the brake value.

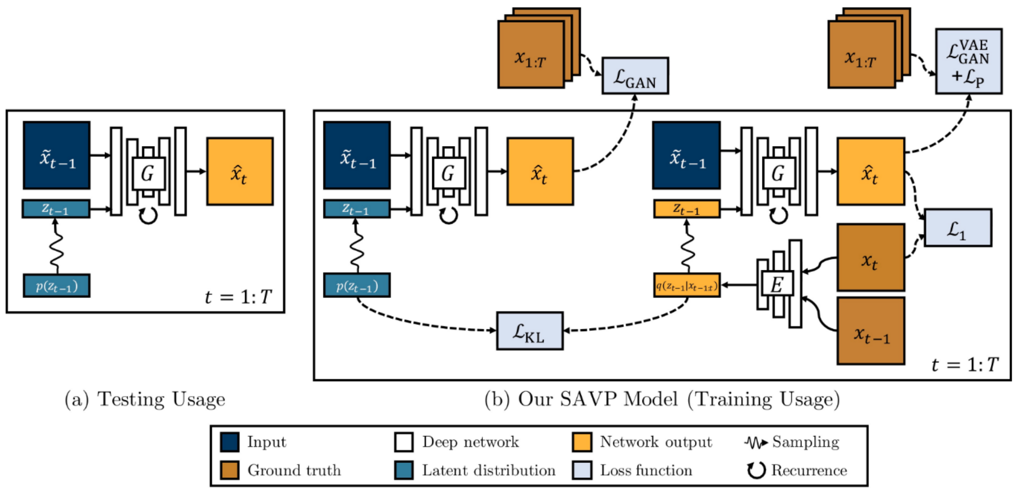

Figure 3: Stochastic Adversarial Video Prediction Network

3.1.1 Stochastic Adversarial Video Prediction Network

I used [24] network architecture and their open source code to generate predicted future frames for the scene in-front of the vehicle giving the current one and previous ones. I changed some hyper parameters to fit my own data-set and train the network on CARLA data-set [9] by giving the network the current frame and three previous frames to generate 4 predicted future frames and then used the up-next four frames in the same scenario from the collected data to be the ground truth.

3.1.2 End-To-End self driving car Network

Convolution neural network is used as the feature extractor followed by fully connected layers to map the input which consists of raw pixels from camera cocoon and the vehicle speed in the current timestamp to the three actions directly, throttle, steering angle and brake. Figure 4 shows our proposed model architecture in which the input image path uses eight convolution layers followed by two fully connected layers. The input vehicle's speed go through two fully connected layers, then the last two fully connected layers which come from the image's path and the speed's path are concatenated with each other then the concatenated layer is followed by three fully connected layers then the output layer. This problem is considered as regression on the continuous control actions, Adam optimizer is used with decayed learning rate as in the following equation 2 starting from 10^{-4}, and decaying rate equals to 0.96. The input image is (200*200*number of stacked images) and outputs directly the three vehicle actions.

Inspired by [9] a conditional imitation learning is used while training the agent as i used four heads to separate each task to a specific head as follows, straight head, left head, right head and finally the follow lane head as shown in Figure 4 there is a high level command controls the four heads and according to this command we take the output from the corresponding head. As shown in [9] this method achieves a better accuracy than inputting the command as a input for the network. Regarding the network which is used in the training I didn't use the network mentioned in [9], I only take the data and the concept from them and implement my own network architecture which is smaller than their network for computation purposes to make the training faster

$$decayed\_learning\_rate = learning\_rate * decay\_rate^{epoch\_number} \tag{2}$$

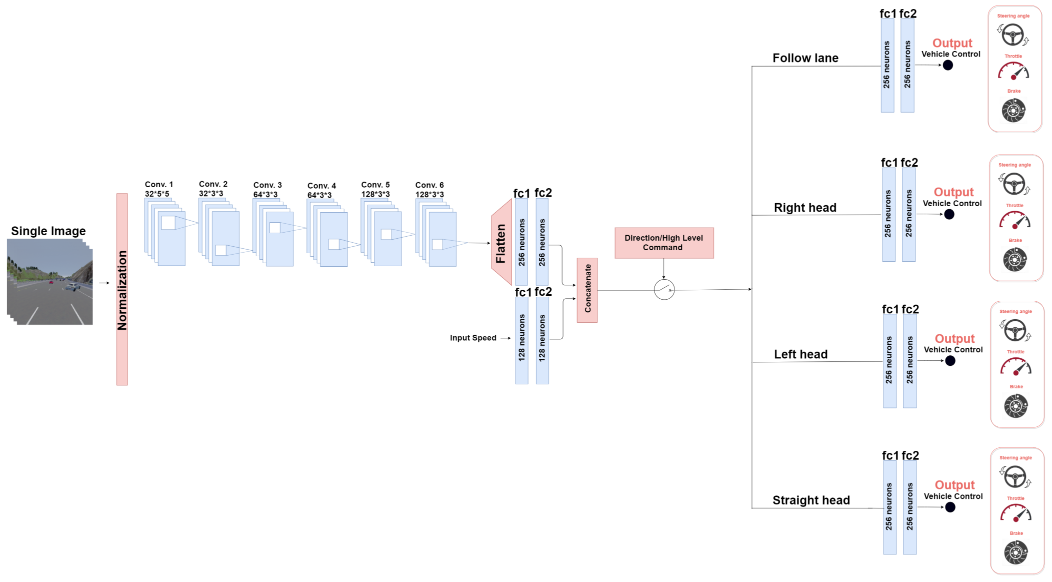

Figure 4: End-to-End self driving car architecture using front camera images

3.2 Training Details

Three agents were trained as follows:

- the first one is considered to be the baseline in-which we will compare with it. The first agent is trained for 50 epochs for 20 hours on GTX 1080 GPU with input size (200*200*4). The input processing as follows, I convert the whole images to gray instead of RGB and stacked the current frame with the previous three frames an the output is the three vehicle actions (throttle, steer and brake).
- The second agent uses the network mentioned in [24], I trained the agent to be able to take four images which are corresponding to the current frame and three previous ones then

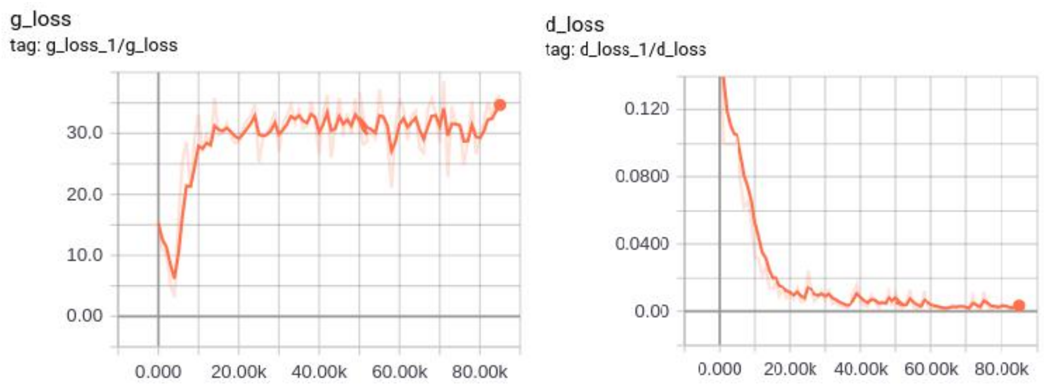

Figure 5: Generator and Discriminator Losses

generate the next four frames, as shown in Figure 6 here is a sample outputs as the first four images from the left side are the input and the following four images from the right side are the output. The training process took 30 hours on GTX 1080 GPU to complete 150 epochs using a constant learning rate equals 10^{-4} and the input and the generated output sizes is (64*64*3) as i trained on small RGB images. As shown in Figure 5 the generator and the discriminator loss versus training steps. One important thing to be noted here, I tried to investigate the reason why the generator loss is increasing and also tried the proposed solution in the main GANS paper which was train the generator twice or three times more than the discriminator but unfortunately it doesn't work for me.

- The third agent is similar to the first one except the network input size becomes (200*200*8) as we provide the current, the previous and the predicted future frames using the second agent. The training process took 23 hours on GTX 1080 GPU to complete 35 epochs using a decayed learning rate equals following equation 2

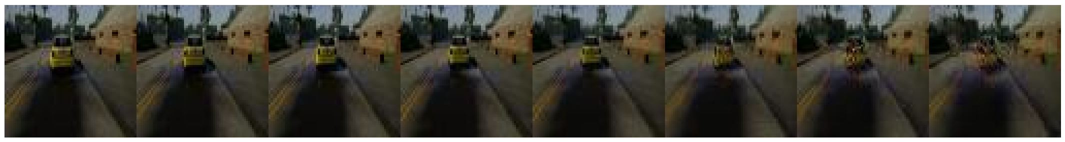

Figure 6: Sample output from the Future prediction Network using [24]

4 Experiments

We use CARLA [10], an urban driving simulator, to corroborate design decisions and evaluate the proposed approach in a dynamic urban environment with traffic. CARLA is an open-source simulator implemented using Unreal Engine 4. It contains two professionally designed towns with buildings, vegetation, and traffic signs, as well as vehicular and pedestrian traffic. Figure 5 provides maps and sample views of Town 1, used for training, and Town 2, used exclusively for testing.

4.1 Experimental Setup

Using CARLA [17], an urban driving simulator, to corroborate design decisions and evaluate the proposed approach in a dynamic urban environment with traffic. CARLA is an open-source simulator implemented using Unreal Engine 4. CARLA version 8.4 is used which contains two professionally designed towns with buildings, vegetation, and traffic signs, as well as vehicular and pedestrian traffic.

The use of the CARLA simulator enables running the evaluation in an episodic setup. In each episode, the agent is initialized at a new location and has to drive to a given destination point, given high-level turn commands from a topological planner. An episode is considered successful if the agent reaches the goal within a fixed time interval. In addition to success rate, we measured driving quality by recording the average distance travelled without infractions (collisions or veering outside the lane).

The two CARLA towns used in our experiments are Town 1 and Town 2. Town 1 is used for training, Town 2 is used exclusively for testing. For evaluation, we used 50 pairs of start and goal locations set at least 1 km apart, in each town.

The benchmark consists of 24 experiments for each CARLA town containing:

- A task for going straight.
- A task for making a single turn.
- A task for going to an arbitrary position.
- A task for going to an arbitrary position with dynamic objects.

Each task is composed of 25 poses that are repeated in 6 different weathers (Clear Noon, Heavy Rain Noon, Clear Sunset, After Rain Noon, Cloudy After Rain and Soft Rain Sunset). The entire experiment set has 600 episodes and took almost 20 hours to be run on each agent.

4.2 Experimental Results

Note that in Table 1 the success rate percentage is calculated as the division between the success episodes number and the total episode numbers on the four tasks and on all different weather conditions. Regarding the reported traveled distance in the last two columns in table 1 I reported them in percentage but in [9] they reported them as absolute distance not relative to the total distance. Table 1 shows comparable results between my baseline and the result in [9] using the shallow network, the differences between my architecture and the shallow one are they used 3 CNN layers but mine have 6 CNN layers and they use dropout and batch-normalization but in my case i didn't use any thing of this, I used only CNN layers and fully connected ones as shown in Figure 3.1. In case of using the full net mentioned in [9] the results become more better than others but i cannot compare with it due to the simple network architecture used in this project, so it is fair enough to compare with their shallow-net.

Regarding the proposed enhancement method using the predicted future frames, it seems a great success as expected, as it makes the agent more robust and could take a concrete decisions not only depending on the current and previous frames but also taking into consideration the future to make a robust decision.

Table 1: Benchmark Results

Experiment	Success rate Town 1	Success rate Town 2	Km Town 1	Km Town 2
Baseline Experiment	49.5%	27%	62.5%	36.5%
Future stacking	54%	31%	66.8%	39%
Original results shallow-net in [9]	46%	14%	0.96 Km	0.42 Km
Original results full-net in [9]	88%	64%	2.34 Km	1.18 Km

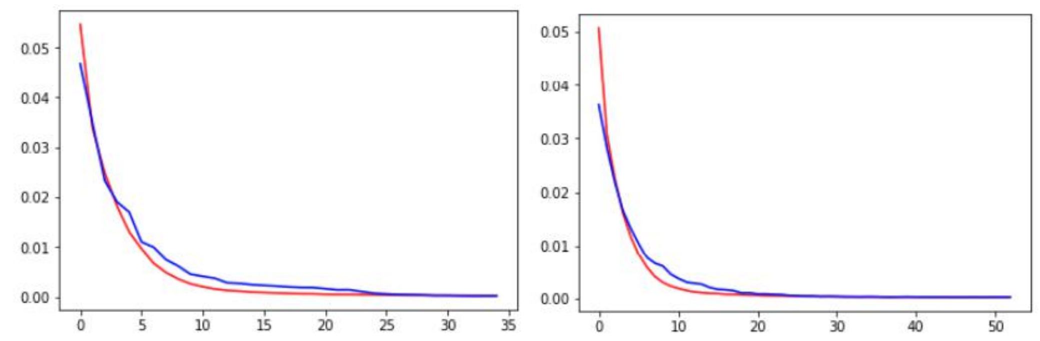

Figure 7: Training loss: on the left part the Future agent loss and on the right part is the base line training loss, The red curves are the loss for training and the blue ones are the validation loss

5 Conclusion

5.1 Summaries the work

The code of the project is on github and it is a well documented and any one could trace the commits history to know the history of adding features one by one.

The code link is : `https://github.com/eslambakr/Future_Imitiation`.

The final weights for baseline model could be found here : `https://drive.google.com/open?id=1NwFh6g7qjSHLMDN963Qzn3xVmUBug3Ck`

The final weights for imitation agent based on future model could be found here : `https://drive.google.com/open?id=1ups8eyg7WYFlua9w8izFa14rCoOM0xq5`

The final weights for future prediction model could be found here : `https://drive.google.com/open?id=19WVB3JB9sDGNJ-KpH2_0JdfYR0vkEO9l`

Deep Learning is contributing greatly in many automotive applications like: autonomous driving, and augmented reality. Fully autonomous driving is one of hot research fields nowadays because of its difficulty. In this work An End-To-End agent was trained on CARLA simulator [17] and was tested on the benchmark developed in [9] and as shown in Figure 7 the loss is decreased on the training data and validation data too. I created my own network architecture as shown in 3.1 and as shown in result it achieve better accuracy than the shallow network which was designed in [9] as shown in Table 1.

Another agent was trained by following [24] to predict the future frames from the current and the previous ones, to make the end to end self driving agent robust enough and although it was trained for 35 epochs only but the baseline agent was trained for 50 epoch, the second agent which make use of the predicted future frames by stacking them with the input achieves a better accuracy than the baseline one as shown in table 1

5.2 Reflection

One of the interesting aspect of the project was i searched a lot to find a network architecture which will be able to predict future frames from the previous ones. For example i read [25] and [26] but i found that it is mainly used for domain adaptation not for prediction however i do believe that we can make modification to make them fit our problem. I tried to use [27] but after investing a lot of time in it, I found it predict the middle frames given the previous and the next one which doesn't fit my application but it will be very useful in other applications like super resolution for images.

One difficult aspect of the project was the timing it seems i have chosen a difficult project which needs more time but i could made it, but if i have more time i could train the agent for more time to be more robust and achieve a better accuracy.

5.3 Improvement

If I have a time I will make a lot of improvements like:

- I will study the affect of increasing the generator loss and trying to solve it as i do believe it will increase the overall accuracy as the imitation agent will be more robustness.
- I will train the agent for more time.
- Adding dropout layers and batch-normalization layers to the imitation network.
- Training [24] using gray images instead of RGB and make use of the computation reduction to make the size of the image bigger for example 200*200 instead of 64*64.
- Make more experiment on the real dataset to prove the effect of taking the future frames in consideration will work on the real data too.

References

[1] O Garcia-Bedoya, S Hirota, and JV Ferreira. Control system design for an automatic emergency braking system in a sedan vehicle. In *2019 2nd Latin American Conference on Intelligent Transportation Systems (ITS LATAM)*, pages 1–6. IEEE, 2019.

[2] Ahmad El Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. End-to-end deep reinforcement learning for lane keeping assist. *arXiv preprint arXiv:1612.04340*, 2016.

[3] Jianbo Lu, Kwaku O Prakah-Asante, and Dimitar Petrov Filev. Smart adaptive cruise control, November 10 2015. US Patent 9,180,890.

[4] Rinku Patel, Christopher Melgar, Andrew Niedert, and Phil Lenius. Enhanced traffic jam assist, May 30 2019. US Patent App. 15/826,973.

[5] Yuji Kodama and Koji Nakatani. Collision avoidance system, November 27 2018. US Patent App. 10/140,867.

[6] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. pages 1–9, Apr 2016.

[7] Hesham M. Eraqi, Mohamed N. Moustafa, and Jens Honer. End-to-end deep learning for steering autonomous vehicles considering temporal dependencies. pages 1–8, Oct 2017.

[8] Zhilu Chen and Xinming Huang. End-to-end learning for lane keeping of self-driving cars. June 2017.

[9] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. Oct 2017.

[10] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[11] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2174–2182, 2017.

[12] Simon Hecker, Dengxin Dai, and Luc Van Gool. End-to-end learning of driving models with surround-view cameras and route planners. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 435–453, 2018.

[13] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, 1(1):33–55, 2016.

[14] Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius Brito Cardoso, Avelino Forechi, Luan Ferreira Reis Jesus, Rodrigo Ferreira Berriel, Thiago Meireles Paixão, Filipe Mutz, et al. Self-driving cars: A survey. *arXiv preprint arXiv:1901.04407*, 2019.

[15] Zhengyuan Yang, Yixuan Zhang, Jerry Yu, Junjie Cai, and Jiebo Luo. End-to-end multi-modal multi-task vehicle control for self-driving cars with visual perceptions. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 2289–2294. IEEE, 2018.

[16] Jinkyu Kim and John Canny. Interpretable learning for self-driving cars by visualizing causal attention. In *Proceedings of the IEEE international conference on computer vision*, pages 2942–2950, 2017.

[17] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[18] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[19] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[20] Will Maddern, Geoffrey Pascoe, Chris Linegar, and Paul Newman. 1 year, 1000 km: The oxford robotcar dataset. *The International Journal of Robotics Research*, 36(1):3–15, 2017.

[21] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, et al. Argoverse: 3d tracking and forecasting with rich maps. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8748–8757, 2019.

[22] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

[23] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018.

[24] Alex X Lee, Richard Zhang, Frederik Ebert, Pieter Abbeel, Chelsea Finn, and Sergey Levine. Stochastic adversarial video prediction. *arXiv preprint arXiv:1804.01523*, 2018.

[25] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[26] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[27] Xiaoou Tang Yiming Liu Ziwei Liu, Raymond Yeh and Aseem Agarwala. Video frame synthesis using deep voxel flow. In *Proceedings of International Conference on Computer Vision (ICCV)*, October 2017.