

Comparative Study of End-to-end Deep Learning Methods for Self-driving Car

Fenjiro Youssef

National School of Computer Science and Systems Analysis (ENSIAS), Mohammed V University, in Rabat 8007, Morocco

E-mail: fenjiro@gmail.com

Benbrahim Houda

National School of Computer Science and Systems Analysis (ENSIAS), Mohammed V University, in Rabat 8007, Morocco

E-mail: Houda.benbrahim@um5.ac.ma

Received: 07 May 2019; Revised: 11 December 2019; Accepted: 14 June 2020; Published: 08 October 2020

Abstract: Self-driving car is one of the most amazing applications and most active research of artificial intelligence. It uses end-to-end deep learning models to take orientation and speed decisions, using mainly Convolutional Neural Networks for computer vision, plugged to a fully connected network to output control commands. In this paper, we introduce the Self-driving car domain and the CARLA simulation environment with a focus on the lane-keeping task, then we present the two main end-to-end models, used to solve this problematic, beginning by Deep imitation learning (IL) and specifically the Conditional Imitation Learning (COIL) algorithm, that learns through expert labeled demonstrations, trying to mimic their behaviors, and thereafter, describing Deep Reinforcement Learning (DRL), and precisely DQN and DDPG (respectively Deep Q learning and deep deterministic policy gradient), that uses the concepts of learning by trial and error, while adopting the Markovian decision processes (MDP), to get the best policy for the driver agent. In the last chapter, we compare the two algorithms IL and DRL based on a new approach, with metrics used in deep learning (Loss during training phase) and Self-driving car (the episode's duration before a crash and Average distance from the road center during the testing phase). The results of the training and testing on CARLA simulator reveals that the IL algorithm performs better than DRL algorithm when the agents are already trained on a given circuit, but DRL agents show better adaptability when they are on new roads.

Index Terms: Self-driving car, deep learning, Imitation Learning, deep reinforcement learning.

1. Introduction to Self-driving Car Levels and Problematics

The self-driving car is a high potential field that knows exponential development in both areas, computer vision, and control systems. It becomes a hot topic and grabs attention with all the hype that AI big players and automakers made around, especially after the great steps performed thanks to the growing research efforts, fueled by the industry, that move this technology from simple prototypes in showrooms and sci-fi movies to real-world cars with almost full autonomy.

The competition is led by IT companies and automobile manufacturers, that are mainly Americans (google Waymo[6], GM[7], Ford[8], Intel Mobileye[9], Tesla[10], Nvidia[11], Uber[12]), Europeans (Volkswagen[13], PSA[14], [15], Renault[16], Daimler-Bosch[17], BMW[18]) and Asians companies (Toyota[19], Honda[20], Mitsubishi, Baidu[21], Didi Chuxing[22], Jingchi[23]), with the ultimate challenge, to conceive autonomous driving systems (ADS) that prevails humans by enhancing safety, decongesting roadways, saving time for users, reducing greenhouse gas emissions, and ensuring mobility for all people, including the disabled and the elderly. These ADS observe and understand their environment by detecting, classifying objects on the road and assessing their speed and direction over time, then predicting and anticipating their behaviors and intents, and finally plan accordingly, the best course of action to perform.

Several modules compose the ADS system, each of which has a specific purpose that helps to ensure that the car stays in its track (drivable path and path delimiters), avoids collisions with obstacles, other vehicles, and pedestrians (object detection), in addition to slowing down or speeding up depending on traffic flow. So, the most important features and building blocks are as following:

- **Adaptive Cruise Control**[24,25] adjusts the car's acceleration, based on the speed of the other vehicles on the road and keep a safe distance from them.
- **Lane-Keep Assist**[26] detects lanes boundaries of drivable paths (even when lanes are non-existent) and generates control actions, in order to leave correct distances from the left and right lines and avoid drifting by detecting unintended deviation towards the lane marking and steering the car back.
- **Road-sign recognition**[27] views signs on the side of the road and applies the related instruction (e.g speed limit, blind spot indicator, ...).
- **Collision Avoidance Systems**[28] detects imminent frontal collisions, alerts the driver to the possible accident ahead and applies the brakes to stop the car in time.
- **Pedestrians Detection**[29] identifies their distances from the car, their postures (walking, running, ...), their behaviors (waving hands, looking direction, ...), their intents, and their faces expressions.
- **Automatic Parking**[30] assesses the possibility of parking a car into a given spot then automatically steers the car to fit that spot.
- **Driver Drowsiness Detection**[31]: uses face expression analysis and lane detection warning to evaluate the driver's attention and ability to drive.

The autonomy of the car depends on the degree of emancipation of human intervention and features coverage. The SAE [32] has divided it into six levels ranging from a driver-operated car (level 0) to a fully driverless car (level 5). **Level 1** providing **driver assistance**, with basic steering, using adaptive cruise control, Lane Keeping Assistance, Parking Assistance, or other acceleration tasks. **Level 2** automated partially the driving tasks by using advanced cruise control, and automated safety actions, like emergency braking and lateral and longitudinal vehicle motion control. From **level 3** we start having full autonomy but under the condition that the human driver will be ready to respond to a request to intervene within some limited time when alerted by the car system. A vehicle with **level 4** automation is capable of fully autonomous driving without any human intervention, except, for areas that are not mapped or while driving in bad weather. For **level 5**, the system is completely autonomous whatever the circumstances.

The last three levels of automation (3, 4, and 5) that address the full autonomy are composed of three main features:



Fig.1. The ADS steps to control the car

The ADS start by addressing localization and routing problem, using GNSS/GPS and navigation data with optimizations algorithm, to choose the best path to reach the driver/passenger destination, then the driverless car needs to have a visual perception of the environment to decide how to behave locally. For that purpose, it uses Deep Learning (DL) and especially convolutional neural network[33] that are trained in a supervised learning mode to extract visual features, and detect/identify roads and obstacles. It begins by collecting and processing sensors data such as camera, radar and LIDAR[34,35] for 3D representation (depth perception), then combining them as an input of the convolutional network, to construct a more accurate and complete perception of the real world.

The next step is planning of actions taking into account the different constraints of the dynamic environment shared with other vehicles and pedestrians, and the feedback control that enhances the robustness of the ADS system. The acquired perception of the environment generated by these modules is then stacked as an input of a fully connected network (another DL network) that will find through training, the best driving policy that determines successful Path Planning and automates decision-making by adapting speed and steering angle while driving. To reach the optimal policy, three methods can be adopted, [1] Supervised learning methods like imitation learning[2], Reinforcement learning methods [3] or a modular approach[36] that combines the two methods beginning by Supervised learning for all detection tasks then feed the resulting data into a Reinforcement learning module as a planner.

In this paper, we focus on the two first approaches and more precisely on the lane-keeping task using a simulation environment and camera's images as the only input of the policy network. For the first one we use Conditional Imitation Learning (COIL) [1], and for the second, we use DQN and Deep Deterministic Policy Gradient (DDPG), which are detailed respectively on the third and fourth chapters. Thereafter, we compare both methods using different experiments, which is described in the fifth chapter, and finally, make conclusions on the usage of such technologies and give an outline of the challenges to come over.

2. Self-driving Using Simulators

To reach an accurate motion planner for autonomous driving using a control policy or cloning expert behavior, we need to verify and assess the control algorithms compliancy, however, driverless car agent cannot be trained in a real environment, since it will be neither affordable in terms of cost, nor acceptable in terms of security and safety (damage

to people, cars, and surroundings). Therefore, most researchers in academic and industry involved in this field always begin by training their agents in a virtual testbed environment to validate their driving strategies. These environments are a great alternative since they allow testing AI car agent algorithms for Reinforcement learning or supervised learning mode without any real risks; and once the policy is good within the defined criteria, then the tests start in ground truth to verify and validate policy learned in the simulation.

For simulators, we have many levels of complexity, starting by the basic ones like Enduro and CarRacing from OpenAI and DeepTraffic a JavaScript simulator made by MIT that are great places to start training AI agents and testing new algorithms, since it doesn't need a lot of hardware resources and time. Followed by more advanced open racing car simulators like TORCS[37] and speed Dreams[38] but they lack numerous basic elements like pedestrians, traffic lights, intersections,.. etc; they also don't offer control over the environment and have limited feedback parameters and options.

Since the goal wasn't only to race on a track but driving agents in a urban environment supported by a physics engine, allowing to sticks as much as possible to ground truth, the choice was restricted to the open-source autonomous driving simulator CARLA [4,5] developed by the Dosovitskiy Computer Vision Center, along with Intel Labs and the Toyota Research Institute and built on top of the UnrealEngine4 game engine. In this paper, we use CARLA simulator for Reinforcement learning and Conditional Imitation Learning benchmark.

CARLA is High-fidelity realistic driving environment that supports camera raw image and provides direct measurements such as forward speed, orientation. The agent has three main Commands Steering angle [-1.0,+1.0], Throttle and Brake [0,+1.0], that are Normalized Float values. We constrain throttling and braking values to prevent over speeding and braking when accelerating by using adequate thresholds:

- Throttle = 0, when:

$$\text{Speed} > V_{\text{Threshold}} \tag{1}$$

- Brake = 0, when:

$$\text{Throttle} > \text{Throttle}_{\text{Threshold}} \tag{2}$$

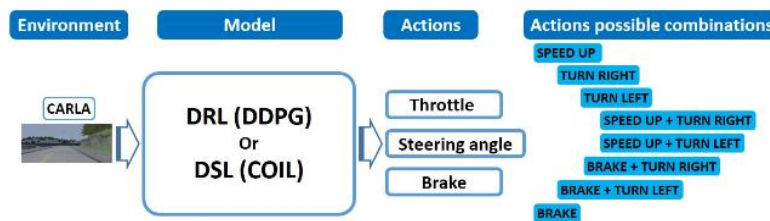


Fig.2. End-to-end models IL and DRL for self-driving car

CARLA is also advanced high resource consumption environment that requires a GPU to run, for the benchmark simulations we use, a dual Xeon processor server, with 32 cores with 32Go of RAM and an Nvidia GPU Geforce GTX 1080Ti (3584 Cuda cores), which allow us to accelerate our benchmark simulations.

For Autonomous Driving systems we have two main architecture approaches, the first one is an end-to-end system that relies on deep neural networks to support the whole chain from the image to the control commands, like imitation learning a supervised learning mode and deep reinforcement learning. The second is a classic modular pipeline that decomposed the driving task into three differentiable subsystems. In our benchmark, we focused on the end-to-end learning algorithms, which behavior will be dissected in the next chapters and compared to see which of them are the most appropriate to solve self-driving complexity.



Fig.3. CARLA Simulator, car in/off road.

3. Supervised Learning Methods

The end-to-end model for self-driving cars trained in supervised mode is based on an imitation learning algorithm [39,40] called Behavioral cloning.

BC was used for driverless cars to replicate driving expert behavior by following the trajectories with higher accuracy. ALVINN[41] (Autonomous Land Vehicle in a Neural Network) was the first endeavor, a fully connected network with 3-layer and a one camera input, followed fifteen years later by DAVE[42] (DARPA Autonomous Vehicle) in 2004 based on a 6 layers convolutional network and an input of two cameras coupled with left and right steering commands, and then came DAVE-2 in 2016, made by Nvidia research team, relying on a 9 layers neural network, including a normalization layer, 5 convolutional layers, and 3 fully-connected layers, using three cameras (left-center-right) and a mean squared error loss. It demonstrates that convolution neural networks are capable of learning the entire line and path tracking task, without the need for a manual decomposition of visual features.

BC cannot infer driving decisions (e.g turn left or right in a crossroad), which need a path planner or human intervention. In this paper, we use the Conditional imitation learning (COIL) algorithm, which is an off-policy learning method and an enhancement of Behavioral cloning (BC)[1] that copy and replicate demonstrations of expert driving behaviors, with the possibility to apply an additional external control. This capability is used to guide the car to take a specific turn at a given intersection.

COIL is a deep policy neural network, a function approximator that works as a regressor to map input camera/Lidar images to sequential control commands “steering” and “acceleration”. It performs well for lane following and off-road obstacle avoidance. Like BC algorithm, COIL allows training a deep neural network “policy” to make decisions mimicking pre-collected expert demonstrations, a dataset of driving trajectories $D = \{\tau^0, \dots, \tau^n\}$, until the network policy reach an optimal parameterization, through the minimization of a loss function, leading to a good imitation performance, by finding the optimal net weights that minimize the difference between the predicted steering angle and speed $(\hat{\theta}, \hat{s})$ and the target angle (θ, s) .

The COIL Neural Network architecture is composed of the Convolutional Network part that allows to catch the visual features that are plugged to a fully connected network that generates the command’s output, see the figure below:

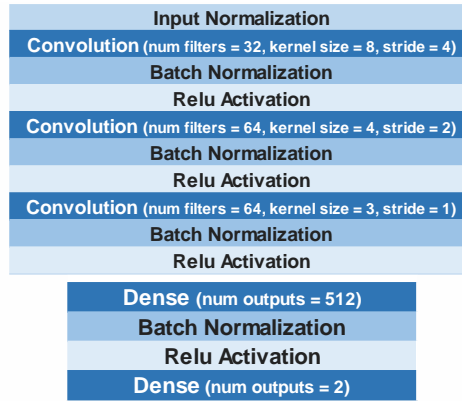


Fig.4. The COIL Network architecture

For supervised learning mode, we have diverse open-source datasets for autonomous driving including complex scenarios, like Kitti[43], Synthia[44], CityScapes[45], and ApolloScape[46].

Knowing that the training and testing are done using CARLA simulator, we create our own dataset, by driving in manual mode, using keyboard keys or mouse movements, and recording the trajectories images, steering angle, and speed. We note (see fig.6. below) that the dominant value for the steering angle and the brake is 0, and for speed we mainly have two values 0 and $0.8 \cdot V_{\max}$ (the speed is capped at $V_{\max} = 30\text{Km/h}$).

The resulting dataset:

$$D = \{\tau^0, \dots, \tau^n\} \quad (3)$$

With τ^i a complete record of a given trajectory, Such as:

$$\tau^i = \{I_k^i, S_k^i, T_k^i, B_k^i\}_{k \in \{1, \dots, n\}} \quad (4)$$

where:

I_k^i : k^{th} image of the i^{th} trajectory

S_k^i : k^{th} steer value of the i^{th} trajectory

T_k^i : k^{th} Throttle value of the i^{th} trajectory

B_k^i : k^{th} Brake value of the i^{th} trajectory

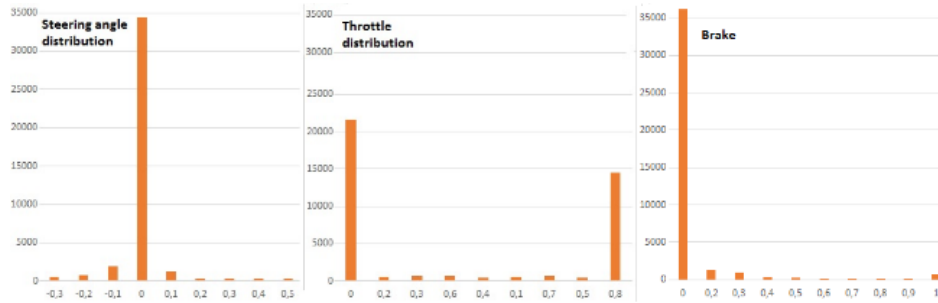


Fig.5. Dataset steering angle, throttle, and brake distributions

The Preprocessing of the dataset images before training the model is extremely important for the correct preparation of data entering the neural network. What is more relevant to the problem to be solved, the results of the neural network will be more accurate. Also, to enhance and optimize the training, we go through several stages of data preparation of the dataset D composed of the labeled images, beginning by lowering the video sampling rate at 10 FPS, to avoid the high similarity and correlation of sequential images, normalizing values of steering angles and speed to $[-1, 1]$, removing the unnecessary part of the image (car's hood, sky, trees ...), blurring the images to smoothen the road lane, balancing the skewed distribution of steering angles to avoid overfitting, since steering angles are almost small with positive values, because the track is mostly straight and most turns are on the right (see histogram figure), therefore we augment our data with its reflection and balance the left and right turns by randomly flipping horizontally the images with equal probability, to remove asymmetry, then we overweight the other angles to get a nearly a Gaussian distribution.



Fig.6. Images processing: cropping and flipping.

Also, due to the correlation of sequential frames, the use of one camera won't respect the i.i.d constraint (independent and identically distributed) and the errors grow in a quadratic way in the length of the horizon, therefore, the input consists of 3 cameras images and 2 floating numbers in the interval $[-1, 1]$, steering angle and speed. Batch Normalization allows using higher learning rates without paying attention to the network weight's initialization.

Experiments show that cropping the image (top and bottom) was a crucial trick that leads to a good performance, avoiding car drifting especially on sharp turns. Followed by resizing the image that reduces the model's parameters and training time by the way, and using image flipping moderate the trend to derive to the left or the right depending on the training track. Also, we notice that training on a track but testing with another different track doesn't perform as well as testing with the same track, which is can be caused by the insufficiency and diversity of the training data or by the overfitting.

Finally, to have a more robust model, it must be able to auto-correct the direction of the car in case of drifting off the road to put it back on the right path, therefore, we feed it with labeled images of car drifting from the center of the lane, with steering angle that goes in the opposite direction, to counter this drift.

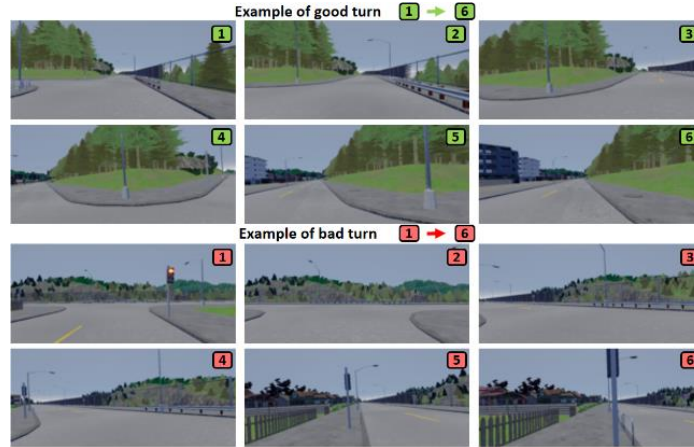


Fig.7. Two scenarios illustrating examples of good and bad bends taken by a car.

The COIL model was trained on CARLA simulator, the model with the lower loss gives the best result with the validation set that we use to prevent overfitting and ensure a better generalization. We then test on Carla simulator the trained model, which performs well.

Supervised Imitation learning including COIL needs great datasets of human expert driving records that cover all kinds of roads, different situations, and driving in various weather conditions. The training phase is very crucial, and the performance of the model depends closely on the availability, diversity, and completeness of the datasets.

4. Reinforcement Learning Methods

Deep reinforcement learning (DRL) are end-to-end algorithms that combine two technologies, deep convolutional neural networks, which are good at extracting visual features and reinforcement learning algorithms which are good at inferring driving policy for action planning, thereby with DRL, we create a single agent that can sense and assess its environment, to act autonomously in the real world. For self-driving cars, we train DRL agents relying on the state input composed from the observed surroundings which are mainly a 2D image representations and distance measurements outputted by cameras, that are processed by a CNN network to detect all the visual features (road, obstacles, vehicles, pedestrians, ...), then the resulting features feed an FCN that generate control signals for the steering wheel, the throttle, and the brakes. DRL is based on Markov Decision Process (full or partial) that uses a reward function as a substitute for the labeled data for supervised learning.

DRL algorithms take actions to maximize the reward function that was designed to incentive the agent to stay in the lane on a track, avoid obstacles and achieve a highest average speed over time, using a simple handy crafted sparse function or inverse reinforcement learning[47] to find a more accurate, complex and continuous form of the reward function, to deal optimally with an environment that varies dynamically and contain objects that behave unpredictably.

The whole DRL is a control policy network whose training is based on the trial-and-error principle that allows self-improvement by balancing exploration vs exploitation using action selection strategies (ϵ -greedy, Softmax, Bayesian, ...). It then uses the acquired knowledge, but also try new actions to discover the optimal way to act throughout navigation. In this paper we will focus on two DRL algorithms, an policy optimization model DDGP[3] and a value optimization model DQN[2].

For the reward function, we construct it in such a way as to encourage speed without exceeding a given threshold and penalize the invasion of other lanes, off-road driving (intersecting the car's bounding 2D box against the road/lane), collisions with obstacles, and steering to prevent excessive turns. The goal is that the car sticks to the center of the road and avoids catastrophic shortcuts that violate the driving rules.

$$\begin{aligned} \text{Reward} = & [\text{capped speed}] - \alpha_1 [\text{intersecting other lanes}] \\ & - \alpha_2 [\text{off road}] - \alpha_3 [\text{Collision}] - \alpha_4 |\text{steer value}| \end{aligned} \quad (5)$$

With α_1 , α_2 , α_3 and α_4 are weights chosen to give more or less importance to a given feature, for example, since collision must be severely punished, we can choose a high value for α_1 (e.g. $\alpha_1=100$), and since intersecting lane is a major issue but not catastrophic, we can choose a medium value ($\alpha_2=5$), for the "off-road" parameter, we choose a higher value ($\alpha_3=10$).

We assess the performance of the DRL algorithms by evaluating the cumulative reward, which indicates the reward gathered by the agent for an entire episode. Being very noisy, we average reward metric and cut negative values to appreciate the ascendance of the agent capacity to drive longer and collect a better reward.

In the next two paragraphs, we go through DRL algorithms DQN and DDPG and their application to the lane-keeping task, and expose the results obtained.

DQN:

DRL algorithms show great results when dealing with discrete and limited action spaces environment using value-based methods such as DQN[2] which was the first application of Q-learning to deep learning, performed by Google DeepMind in 2015, succeeding to play 2600 Atari games at expert human level. DQN that is maximizing over discrete action selection (maximum Q value output) need to adapt to the self-driving car in real-world that evolve in continuous action space, by performing a space discretization at the expense of the requirement of fine control.

During the tests, we notice that this type of methods is not a good choice since the cumulative reward did not soar and stay always in negative values (see fig.8. below).

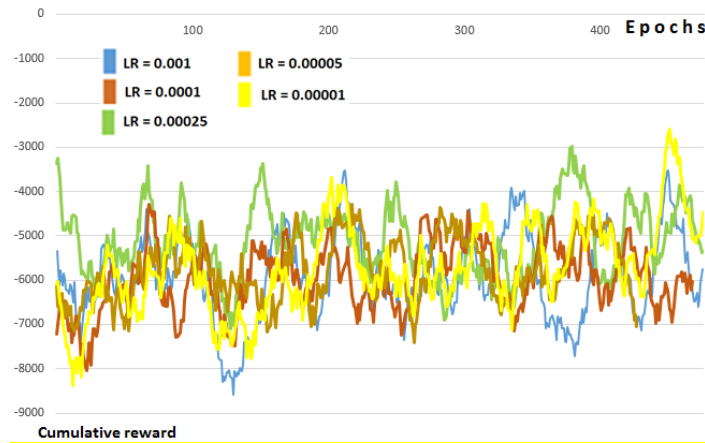


Fig.8. Evolution of the average of DQN cumulative reward with different learning rates

DDPG:

The state space in the real world is extremely complex and action space is continuous, therefore, fine control is required, which is possible using policy gradient or Actor-critic algorithms. This last one is a hybrid of policy based (actor = policy gradient) and value-based methods (critic = Q function).

In our benchmark, we use Deep Deterministic policy gradient algorithm DDPG[3] as an end-to-end off-policy RL learning algorithm for continuous action space, based on actor-critic neural network, that predict the action for the current state and minimize separately two losses L_{Actor} and L_{Critic} and learn a **deterministic** target policy. DDGP re-use DQN tricks:

- **Experience replay buffer** to solve correlated data issues
- **Target Network**, make copies (Q', μ') of the Actor and Critic networks (Q, μ) and soft updates to enable training stability:

$$\theta^{Q'} \leftarrow \tau \cdot \theta^Q + (1 - \tau) \theta^Q \quad (6)$$

$$\theta^{\mu'} \leftarrow \tau \cdot \theta^\mu + (1 - \tau) \theta^\mu \text{ with } \tau \ll 1 \quad (7)$$

- **Exploration** by adding noise to actor actions:

$$\mu_{Exploration}(s_t) = \mu_\theta(s_t) + N_t \quad (8)$$

For DDPGG, the architecture of the actor and critic neural networks are the same for the visual features extraction, which uses three blocks each of which contain a convolutional layer, a batch normalization layer, and a Relu activation layer, see the fig.9 below.

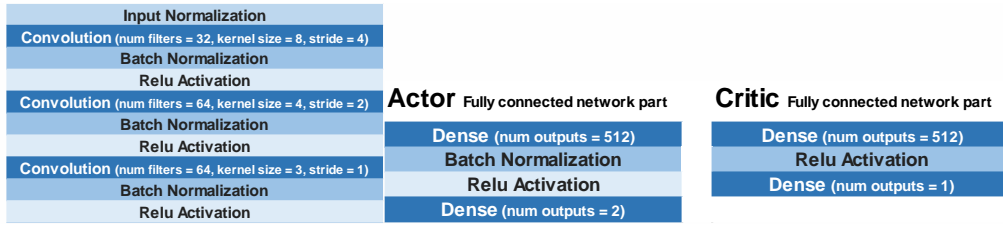


Fig.9. The DDPG Network architecture

DDPG has an actor-critic architecture with two different learning rate (LR), and during the training, we choose the actor’s LR (e.g. 10^{-4} vs 10^{-5}) lower than the critic’s LR since we want the actor to achieve many actions before being assessed for his performance by the critic.

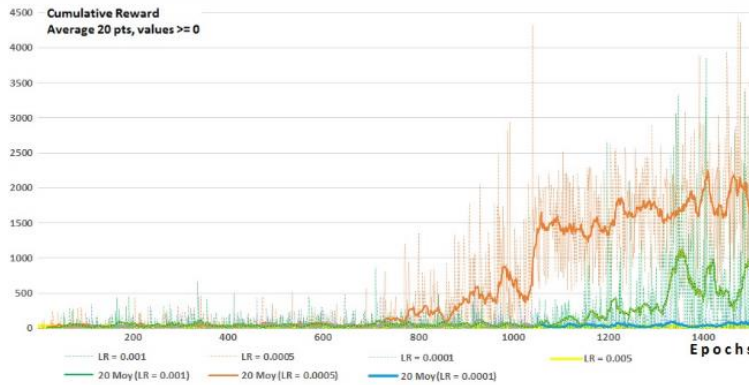


Fig.10. Evolution of the DDPG cumulative reward (value > 0) for different learning rates values

By varying the LR, we notice that the impact on total reward was significant, for LR actor value from ($5 \cdot 10^{-3}$, 10^{-3} , $5 \cdot 10^{-4}$, 10^{-4}) and critic LR ($5 \cdot 10^{-4}$, 10^{-4} , $5 \cdot 10^{-5}$, 10^{-5}). With lower LR, the total reward starts his ascendance earlier and reach greater values (see a test of the trained model on Youtube[48]).

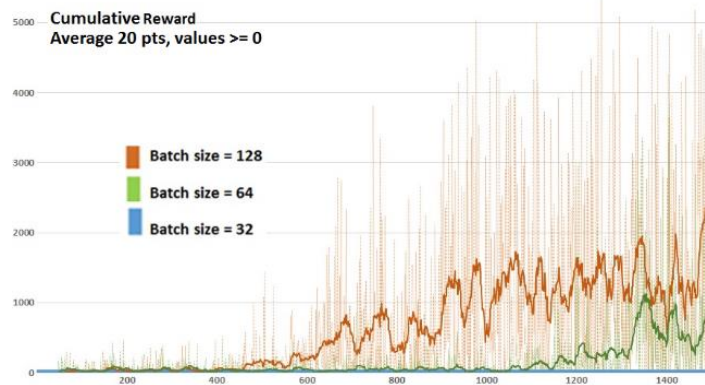


Fig.11. Evolution of the DDPG cumulative reward (values > 0) for different batch size values

For the network hyperparameters tuning, we found that bigger batch size (the number of images used for each gradient descent update) enhance updates stability, however, it leads to a slower training. A smaller number of epochs (the number of passes through the experience buffer during gradient descent) promotes more stable updates, but slower training. A larger Number of epochs value is acceptable when the batch size is large. For the Learning rate, a larger value can increase the training speed but may cause the expected rewards not to convergence if it is too large. The trade-off between exploitation and exploration was insured by the Epsilon Greedy strategy discount factor. We use Huber loss (instead of MSE loss) as it takes smaller increments, which bring more stability and good convergence. Another thing is I found out that slightly updating the target network at every time step (soft update) worked better for me than updating it at each N time steps (hard update).

5. Comparison of SL & RL Approaches

In this chapter we are comparing SL approach (Conditional imitation learning) vs RL approach (deep

reinforcement learning) by focusing on some key indicators that are shared between the 2 models, the first one is related to machine learning domain which is the Loss function, and two others that are specific to self-driving car domain which are “the episode duration before a crash” and “the average distance from the road center” that both evaluate the quality and stability of the self-driving algorithm.

In this comparison, we use DDGP RL algorithm, with the best configuration tested, namely, a learning rate of 0.0005 and a batch size of 128. For COIL we used the same learning rate of 0.0005.

Comparing the basics

DRL and IL end-to-end models are simple, attractive and very appealing approaches for a self-driving car, below a comparison between the basic principles of each technology:

Table 1. IL vs DRL

Imitation Learning: COIL	Deep Reinforcement Learning: DDPG
Learning Expert behaviors	Self-learning through trial and error
Cannot exceeds expert skills	Can innovates and finds new tricks
Trains offline and requires pre-collected labeled Dataset.	Trains online and requires access to the environment.
Compares predicted values with labels to improve the actions of the learning agent.	Optimizes the cumulated reward to get the best scoring for its actions.
Huge dataset required	reward function is sparse or difficult to specify
Demonstrations must cover the entire state space	Play many episodes to explore and exploit the best scenarios
No long term planning	Learning optimized action planning
fast learning	slow learning

But both technologies suffer from some limitations, especially for self-driving car context that requires high accuracy and performance since a 1-step deviation can lead to catastrophic error.

In the following, we are comparing RL and BC based on three indicators:

- The Loss
- The episode duration before a crash
- The average distance from the road center

Loss

In SL, the goal is to minimize the loss function through training, using gradient descent or other optimization algorithms, to decrease the difference between the labels and predicted values. Therefore, we need a huge amount of pre-collected labeled as a dataset to maximize the accuracy of our resulting model.

During the training, multiple epochs are performed with data mini-batches from the same initial dataset, to tweak the neural network weights, so as they fit the dataset labels, and since it’s done in an offline mode, the actions do not affect the environment, and the loss decreases as the training progress, and this is exactly what we notice for the COIL model (see fig.12 below).

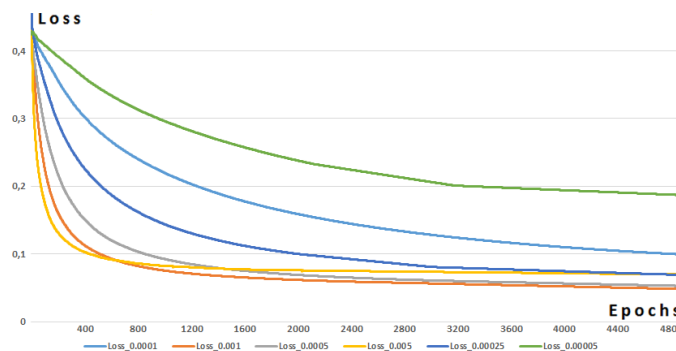


Fig.12. The evolution of COIL loss function for different learning rates

While in RL, the goal is to improve the cumulated reward; and we notice that the total reward per episode can increase without necessarily decreasing the loss during the training. This is due to the changes that affect the policy every N episodes (depending on the update frequency), also the action-selection strategy (exploration vs exploitation) adopted introduces another stochastic change, and then different actions are taken for each episode, which modifies the environment and so the input data. Therefore, the loss cannot capture how good the policy is since the initial input data

has changed, as a result, in DRL, the loss may not converge as the data keeps changing when the policy is updated. This was confirmed during the training of the DDPG model see Fig.13 below.

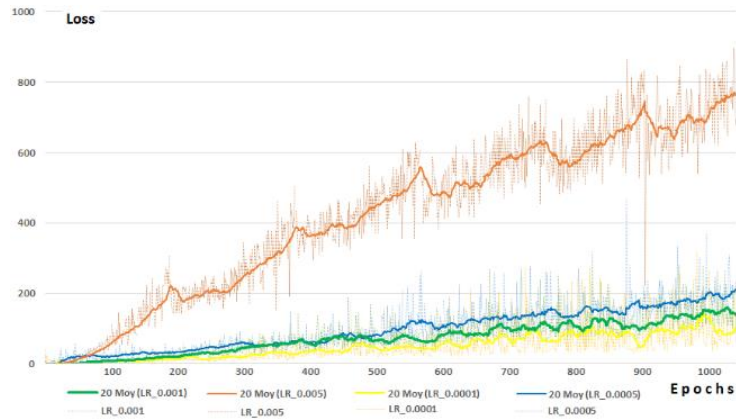


Fig.13. Evolution of the Loss for DDPG algorithm for different learning rates values

Episodes duration (before a crash)

We launch a series of tests (100 episodes) whose duration was capped to 180 seconds with both Reinforcement Learning DDGP and Imitation learning COIL algorithms, trained on the Town1 of Carla simulator, with the same environment conditions.

We notice (see fig.14. below) that self-driving in Town1 circuit, using COIL model trained on Town1, give the best results, however, when the COIL agent drive on Town2 circuit, using the same COIL model (trained on Town1), it gives the worst results. The drop on performance is very clear, which is due to distribution mismatch between training dataset and testing dataset, and also because some state spaces were not covered by the expert trajectories.

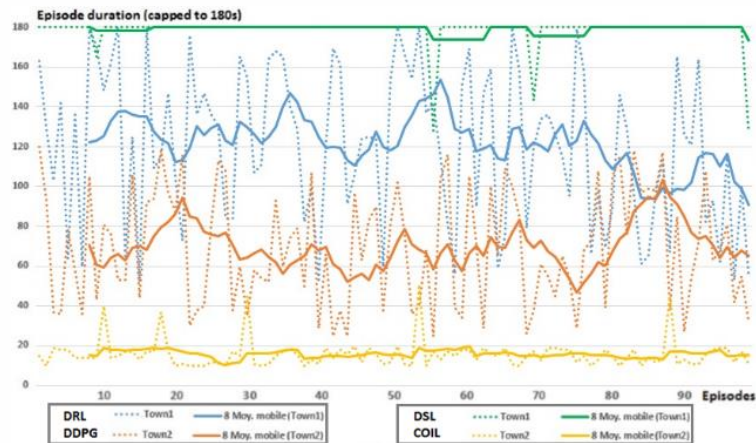


Fig.14. Episodes duration before a crash for DDPG and COIL algorithm tested on Town1 and Town2

We also notice that RL model trained on Town1 and tested on Town1, give better results than RL model trained on Town1 and tested on Town2, but the drop on performance is not so great comparing to the GAP noted for the BC model.

We can deduce that COIL model is better than RL model for self-driving car on circuits that are already trained on, but perform weakly when the circuit is different, on the other hand, RL model proves its ability to adapt to new situations better than the BC model, but the results remain unsatisfying and perfectible.

The average distance from the road center

Through the experimentation performed for the two approaches, COIL agent seems to stick more smoothly to the curve of the road than RL agent, when driving on a road that he was already trained on, and shows better stability during the driving test, since the average distance from the center of the road remains small. On the other hand, it has the worse performance when trained on a circuit and tested on another different one, which raises another time the problem of Data distribution and coverage of the action space by the training Dataset.

On the other side, we notice that RL agent trained on a given circuit and tested on the same one performs better than the agent trained on a circuit and tested on another one. RL agent also suffers from instability and sometimes drives the car zigzag along the way.

We can note (see fig.15 below) that COIL agent behaves better than RL agent does, when it is already trained on the test circuit, but its performance drop when the testing road is new or quite different.

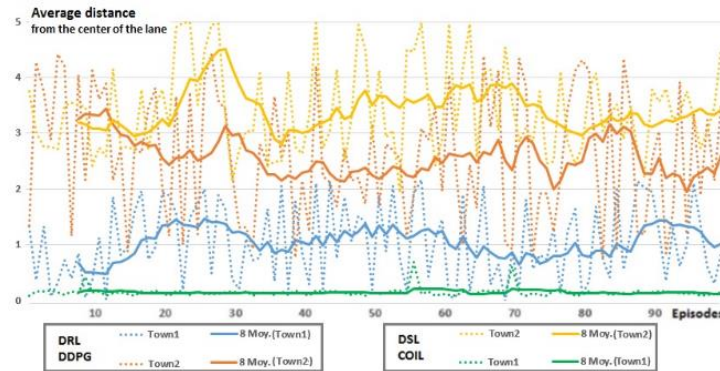


Fig.15. The average distance from the center of the lane for DDPG and COIL algorithm tested on Town1 and Town2

Summary of the comparative study

The tests and comparison show that DRL DDPG and IL COIL perform well, but IL outperforms DRL when dealing with known roads, however, DRL adapts better for unknown roads. IL models cannot reach a good accuracy, unless it trained on a huge dataset with different kind of roads, in different conditions and scenarios, and DRL need training a long time to enhance its accuracy through trial and error, exploration/exploitation and designing a practical reward function manually or through inverse reinforcement learning[47,49].

As a summary, we can say that both technologies have some limitations; consequently, the actual level cannot perform safety driving and need more improvements and research efforts. Some implementations propose to combine the two algorithms, by training the policy network using behavioral cloning at the beginning and then switch to deep reinforcement learning algorithm with the same network to enhance the agent skills and improve the resulting policy. Others use modular pipeline approach with separate deep learning modules trained in a supervised learning mode to assess and recognize the environment elements (traffic signs, traffic lights, road marking, lane split and merge, ...), then put all these modules output information as an input to a reinforcement learning finale module, that is used as a path planner.

6. Conclusion

The actual driverless car technology has come to a high level of accuracy and reliability, thanks to the great breakthrough in deep imitation learning and reinforcement learning, which made it possible in the real-world. Tesla cars and google Waymo are good examples. This comparative study has adopted a methodology based on training and testing both algorithms on the same High-fidelity realistic driving environment CARLA with the same parameters and the same metrics, which allowed us to understand the strengths and weaknesses of both technologies in the context of Self-driving car. Throughout this benchmark, Imitation learning has in fact, shown great results for deja vu situations and Reinforcement learning has the capacity to deal with new situations, this conclusion based on the training and testing results, led us to the next deduction, that the combination of both algorithms, would be a good opportunity to leverage the strength of both of them. Therefore, the learning process of an AI agent can begin by imitation learning to quickly acquire the expert's skills, then learns to generalize and adapt to new environments, using Reinforcement learning algorithms. Also, this approach can help to get rid of the weaknesses of Reinforcement learning like bad initialization, slow learning, credit assignment, however, it cannot solve all the limitations like dealing with new complex tasks, partial observability or generalization, which opens the door for opportunities of progress, and great challenges to overcome, by discovering new ideas and breakthroughs. The goal is to make an ADS system more accurate and safer, and get the domain of transportation to another level.

References

- [1] M. Bojarski *et al.*, "End to End Learning for Self-Driving Cars," *arXiv:1604.07316 [cs]*, Apr. 2016.
- [2] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.
- [3] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *arXiv:1509.02971 [cs, stat]*, Sep. 2015.
- [4] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An Open Urban Driving Simulator," *arXiv:1711.03938 [cs]*, Nov. 2017.
- [5] "CARLA Simulator." [Online]. Available: <http://carla.org/>.

- [6] “Waymo,” *Waymo*. [Online]. Available: <https://waymo.com/>.
- [7] “Cruise Automation.” [Online]. Available: <https://getcruise.com/>.
- [8] “Looking Further,” *Ford Corporate*. [Online]. Available: <http://corporate.ford.com/innovation/autonomous-2021.html>.
- [9] “Mobileye | Autonomous Driving & ADAS (Advanced Driver Assistance Systems),” *Mobileye*. [Online]. Available: <https://www.mobileye.com/>.
- [10] “Autopilot.” [Online]. Available: <https://www.tesla.com/autopilot>.
- [11] “Autonomous Car Development Platform from NVIDIA DRIVE AGX,” *NVIDIA*. [Online]. Available: <https://www.nvidia.com/en-us/self-driving-cars/drive-platform/>.
- [12] “Uber Advanced Technologies Group,” *Uber Advanced Technologies Group*. [Online]. Available: <https://www.uber.com/info/atg/>.
- [13] “Autonomous Driving | Innovation | Volkswagen Australia.” [Online]. Available: https://www.volkswagen.com.au/content/vw_pkw/importers/au/en/why-volkswagen/innovation/autonomous-driving.html.
- [14] “Groupe PSA’s safe and intuitive autonomous car tested by the general public,” *Groupe PSA*. [Online]. Available: <https://www.groupe-psa.com/en/newsroom/automotive-innovation/voiture-autonome-2/>.
- [15] “Autonomous car & driving : vehicle automation,” *Groupe PSA*. [Online]. Available: <https://www.groupe-psa.com/en/story/en-route-vers-la-voiture-autonome/>.
- [16] “Autonomous vehicle and autonomous driving - Groupe Renault.” [Online]. Available: <https://group.renault.com/en/innovation-2/autonomous-vehicle/>.
- [17] “Autonomous Driving - CASE,” *Daimler*. [Online]. Available: <https://www.daimler.com/case/autonomous/en/>.
- [18] “Autonomous Driving – five steps to the self-driving car.” [Online]. Available: <https://www.bmw.com/en/automotive-life/autonomous-driving.html>.
- [19] T. M. CORPORATION, “Toyota Research Institute Introduces Next-Generation Automated Driving Research Vehicle at CES | Corporate | Global Newsroom,” *Toyota Motor Corporation Official Global Website*. [Online]. Available: <https://global.toyota/en/newsroom/corporate/20564649.html>.
- [20] “Mobility Meet the Autonomous Work Vehicle Prototype.” [Online]. Available: <https://www.honda.com/mobility/Autonomous-Work-Vehicle>.
- [21] “Apollo.” [Online]. Available: <http://apollo.auto/>.
- [22] “DiDi Labs - Intelligent Transportation Technology & Security.” [Online]. Available: <http://www.didi-labs.com/>.
- [23] “WeRide.ai.” [Online]. Available: <https://www.weride.ai/>.
- [24] W. Pananurak, S. Thanok, and M. Parnichkun, “Adaptive Cruise Control for an Intelligent Vehicle,” 2009, pp. 1794–1799.
- [25] İ. Kılıç, A. Yazıcı, Ö. Yıldız, M. Özçelikors, and A. Ondoğan, “Intelligent adaptive cruise control system design and implementation,” in *2015 10th System of Systems Engineering Conference (SoSE)*, 2015, pp. 232–237.
- [26] Q. Zou, H. Jiang, Q. Dai, Y. Yue, L. Chen, and Q. Wang, “Robust Lane Detection from Continuous Driving Scenes Using Deep Neural Networks,” *arXiv:1903.02193 [cs]*, Mar. 2019.
- [27] P. Sermanet and Y. LeCun, “Traffic sign recognition with multi-scale Convolutional Networks,” in *The 2011 International Joint Conference on Neural Networks*, San Jose, CA, USA, 2011, pp. 2809–2813.
- [28] D. J. Phillips, J. C. Aragon, A. Roychowdhury, R. Madigan, S. Chintakindi, and M. J. Kochenderfer, “Real-time Prediction of Automotive Collision Risk from Monocular Video,” *arXiv:1902.01293 [cs]*, Feb. 2019.
- [29] D. Tomè, F. Monti, L. Baroffio, L. Bondi, M. Tagliasacchi, and S. Tubaro, “Deep convolutional neural networks for pedestrian detection,” *Signal Processing: Image Communication*, vol. 47, pp. 482–489, Sep. 2016.
- [30] “(PDF) Real-time image-based parking occupancy detection using deep learning.” [Online]. Available: https://www.researchgate.net/publication/323796590-Real-time_image-based_parking_occupancy_detection_using_deep_learning. [Accessed: 31-Mar-2019].
- [31] S. Park, F. Pan, S. Kang, and C. D. Yoo, “Driver Drowsiness Detection System Based on Feature Representation Learning Using Various Deep Networks,” in *Computer Vision – ACCV 2016 Workshops*, vol. 10118, C.-S. Chen, J. Lu, and K.-K. Ma, Eds. Cham: Springer International Publishing, 2017, pp. 154–164.
- [32] “SAE J 3016-2018 - Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles.” [Online]. Available: <https://webstore.ansi.org/Standards/SAE/SAE30162018?source=blog>. [Accessed: 31-Mar-2019].
- [33] Salman Khan, Hossein Rahmani, Syed Afaq Ali Shah, Mohammed Bennamoun, *A Guide to Convolutional Neural Networks for Computer Vision*. .
- [34] “(PDF) LiDAR-Video Driving Dataset: Learning Driving Policies Effectively.” [Online]. Available: https://www.researchgate.net/publication/327762708-LiDAR-Video_Driving_Dataset_Learning_Driving_Policies_Effectively. [Accessed: 31-Mar-2019].
- [35] V. De Silva, J. Roche, and A. Kondo, “Robust Fusion of LiDAR and Wide-Angle Camera Data for Autonomous Mobile Robots,” *Sensors*, vol. 18, no. 8, p. 2730, Aug. 2018.
- [36] “[1604.06915] On the Sample Complexity of End-to-end Training vs. Semantic Abstraction Training.” [Online]. Available: <https://arxiv.org/abs/1604.06915>.
- [37] B. Wymann, C. Dimitrakakis, A. Sumner, E. Espie, and C. Guionneau, “TORCS: The open racing car simulator,” p. 5.
- [38] “(PDF) Autonomous Drifting Control in 3D Car Racing Simulator,” *ResearchGate*. [Online]. Available: https://www.researchgate.net/publication/328007272-Autonomous_Drifting_Control_in_3D_Car_Racing_Simulator. [Accessed: 31-Mar-2019].
- [39] Y. Duan *et al.*, “One-Shot Imitation Learning,” *arXiv:1703.07326 [cs]*, Mar. 2017.
- [40] Y. Pan *et al.*, “Agile Off-Road Autonomous Driving Using End-to-End Deep Imitation Learning,” *arXiv:1709.07174 [cs]*, Sep. 2017.
- [41] D. A. Pomerleau, “ALVINN: An Autonomous Land Vehicle in a Neural Network,” in *Advances in Neural Information Processing Systems 1*, D. S. Touretzky, Ed. Morgan-Kaufmann, 1989, pp. 305–313.

- [42] "DAVE: Autonomous Off-Road Vehicle Control using End-to-End Learning." [Online]. Available: <https://cs.nyu.edu/~yann/research/dave/index.html>. [Accessed: 31-Mar-2019].
- [43] "(PDF) Vision meets robotics: the KITTI dataset." [Online]. Available: https://www.researchgate.net/publication/258140919_Vision_meets_robotics_the_KITTI_dataset. [Accessed: 31-Mar-2019].
- [44] G. Ros, L. Sellart, J. Materzynska, D. Vázquez, and A. M. López, "The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3234–3243, 2016.
- [45] M. Cordts *et al.*, "The Cityscapes Dataset for Semantic Urban Scene Understanding," *arXiv:1604.01685 [cs]*, Apr. 2016.
- [46] X. Huang, P. Wang, X. Cheng, D. Zhou, Q. Geng, and R. Yang, "The ApolloScape Open Dataset for Autonomous Driving and its Application," *arXiv:1803.06184 [cs]*, Mar. 2018.
- [47] A. Y. Ng and S. Russell, "Algorithms for Inverse Reinforcement Learning," *ICML '00 Proceedings of the Seventeenth International Conference on Machine Learning*, May 2000.
- [48] *RL DDPG agent driving in the CARLA Simulator - YouTube*. .
- [49] S. Sharifzadeh, I. Chiotellis, R. Triebel, and D. Cremers, "Learning to Drive using Inverse Reinforcement Learning and Deep Q-Networks," *arXiv:1612.03653 [cs]*, Dec. 2016.

Authors' Profiles



Fenjiro Youssef was born on 21 December 1978. He received the Master of degree in Industrial Computer Science from ENSEM-INPL Nancy, France, in 2001.

In 2002, he joined Maroc Telecom as an engineer responsible of Internet platforms. Then became the head of internet IT in 2006, and project manager officer in 2012. In 2016, he enrolls for a doctorate at Mohamed V University, at the IT department of ENSIAS engineering school in Rabat, Morocco, with Mrs. Benbrahim Houda as thesis supervisor.



Mrs. Benbrahim Houda received a Ph.D. in Computer Science from Portsmouth University 2008, UK. She joined Mohamed V University in 2008, as a teacher at ENSIAS engineer school, Rabat, Morocco, where she is teaching (Data Mining, Artificial Intelligence, Data Analysis, and Statistics) and supervising researchers in these fields.

How to cite this paper: Fenjiro Youssef, Benbrahim Houda, "Comparative Study of End-to-end Deep Learning Methods for Self-driving Car", *International Journal of Intelligent Systems and Applications(IJISA)*, Vol.12, No.5, pp.15-27, 2020. DOI: 10.5815/ijisa.2020.05.02