End-to-End Deep Neural Network Design for Autonomous Vehicle Navigation

Dao Minh Quan, **Lanza** Davide

Control and Robotics Master (CORO)

École Centrale Nantes

Supervisor

Vincent **Fremont** †

† École Centrale Professor,

LS2N ARMEN

Group Project Master 1

CORO – IMARO Advanced Robotics

June 27, 2019

Abstract

The classical approach to autonomous vehicle navigation is comprised by the three steps of mission planning, path planning and trajectory planning. Even though the first one of those can be implemented efficiently with a graph planning algorithm, the realization of the other two remains an open question.

Use Convolutional Neural Networks (CNN) can successfully avoid the problems that arise from the classical approach, allowing a mapping from images obtained by a front-facing camera to steering angle by observing human expert drivers.

Starting from NVIDIA's DAVE-2 idea, the research in the area included many architectures that provided successful results, but with different drawbacks. In this project we propose to solve them (like the vanishing gradient problem of regression approaches) using an existing architecture and adapting it to a classification problem.

The main goal is to develop and End-to-end architecture able to completely learn how to drive a car (calculate steering angle and velocity) since DAVE-2 and the other frameworks presented in literature just predict the steering angle. A geometric path will be learned so that it can be later timestamped providing a velocity-profile.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Classic motion planning

Autonomous vehicles design is a core topic in academic and industry research, due to its numerous advantages like safety improvement, traffic and emission reduction, and unsupervised task completion. Motion planning methods are developed in order to perform path planning, obstacles avoidance and best trajectory generation. A range of planning approaches have been proposed in the literature, with the aim of ensure safety, comfort and efficiency. The general motion planning work-flow for autonomous on-road driving normally consists of three phases: finding a path, searching for the safest manoeuvre and determining the most feasible trajectory. As suggested in [1], it is possible to identify four hierarchical classes: (a) route planning, (b) path planning, (c) manoeuvre choice and (d) control planning (that here will be named trajectory planning in coherence with [2]), as shown in the flowchart at Figure 1.1. If the route planning is concerned only with finding the optimal route from two locations (sometimes considering real-time traffic information), path, manoeuvre and trajectory planning take into account vehicular dynamics, obstacles, road geometry and traffic interactions [2].

Planning hierarchy Following the definitions in [2], a path is a geometric trace that the vehicle should follow without colliding with obstacles, in order to reach its destination. The path-planning is therefore to compute that geometric trace starting from two fixed configurations. The resulting path has to be planned in order to ensure feasibility to each configuration and state along it, that is, ensuring collision avoidance and adherence to a set of motion constraints like lane boundaries and traffic rules.

A higher level motion characterization for vehicles is the manoeuvre[1], and it is related to its position and speed. A manoeuvre is called nominal if it can be performed in safety and it is adherent to the manoeuvres constraints (like traffic or other rules). A manoeuvre planner, starting from the path specified from path planning, returns the best high-level decision for the vehicle.

The trajectory is presented in [2] as a sequence of states visited by the vehicle, parameterized by time and (possibly) velocity. Trajectory planning is a real-time planning concerned by the transitions from a feasible state to another, satisfying the kinematic constraints (based on vehicle dynamics and navigation comfort), street boundaries, traffic rules, and obstacles avoidance.

[1] Examples of manoeuvres could be 'going straight', 'turning', 'overtaking' etc...

During a planning cycle the path planner module, starting from vehicle's current location, generates a set of trajectories. The trajectory planner module action is scheduled at regular timings, depending on the frequency of fresh sensor data retrieval. The flowchart of the planning process is shown in Figure 1.1

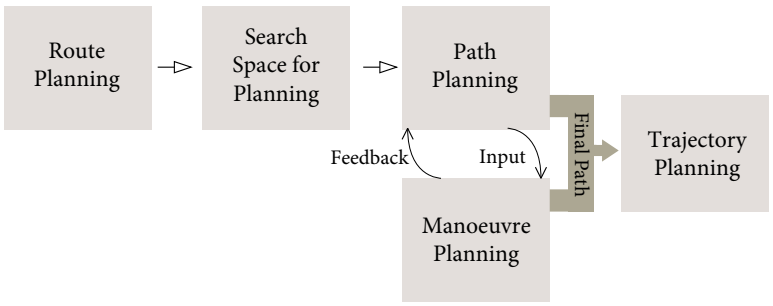


Figure 1.1: Flowchart for planning modules

Planning flow The planning process can be divided into incremental approaches [2] in order to find the best sequence of state transitions allowing information reuse from previous searches. As shown in Figure 1.1, a route is chosen from the route planner, then the path search is initialized and start to work as input for the manoeuvre planner, that will change it until a final path is obtained (the "input-feedback cycle" in the figure).

So, the planning flow it can be divided into three levels [2]: find the best geometric paths for the vehicle to follow, then the best manoeuvres to perform, and finally the best trajectory to follow through the optimization of a geometric curve (according to the constraints given).

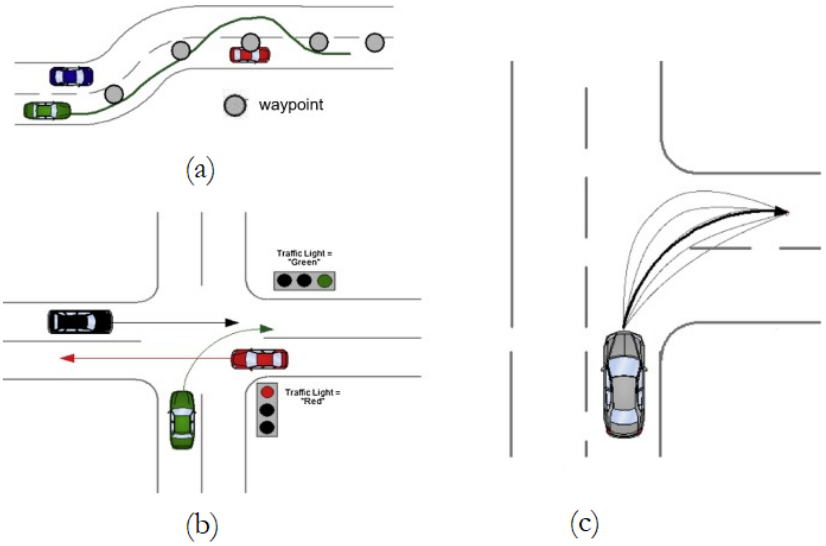


Figure 1.2: Classical path plannig example: (a) path planning, (b) manoeuvre planning and (c) trajectory planning. [2]

Using as example the one illustrated in [2] (Figure 1.2) it is possible to show the different process dynamics. Path planning is illustrated in Figure 1.2a, where the vehicle follows a sequence of waypoints generated by route planner and then constructs the geometric path. In

order to have a path that is obstacle-free the vehicle needs to consider how other vehicles are moving in the road, so, it performs the manoeuvre planning (an example is shown in Figure 1.2b). The higher-level decisions depend on the geometric path generated before, because the manoeuvre planner uses as reference the input waypoints. After the waypoints and the proper manoeuvre are finalized, the trajectory planner computes the best trajectory that connect the determined waypoints (an example is shown in Figure 1.2c).

1.1.1 Planning techniques

Route planning techniques The road environment has to be represented in order to allow the queries of the path planner. In other words, the physical space has to be projected in a configuration space, or in a state space. The sensors data from the vehicle, interpolated with the information contained in the digital map of the environment, are used in order to discretize the physical space while the vehicle is moving. This discretization creates the working domain for the path planner, so it must efficiently dense and expressive, trading off adequate resolution and small computational time. In Table 1.1 are shown four methods for search space generation, illustrated in Figure 1.3.

Voronoi Diagrams	*Advantages*: – Completeness. – Maximum distance from obstacles *Disadvantages*: – Limited to static environments – Discontinuous edges
Occupancy Grids Cost Maps	*Advantages*: – Fast discretization – Small computational power *Disadvantages*: – Problems with vehicle dynamics – Errors in the presence of obstacles
State Lattices	*Advantages*: – Efficient but computationally light – Pre-computation of edges is possible *Disadvantages*: – Problems with curvature – Restrict motion – Difficulties with evasive manoeuvres
Driving corridors	*Advantages*: – Continuous collision free space *Disadvantages*: – Computational cost – Constraints on motion

Table 1.1: Comparison of search space for planning. [2]

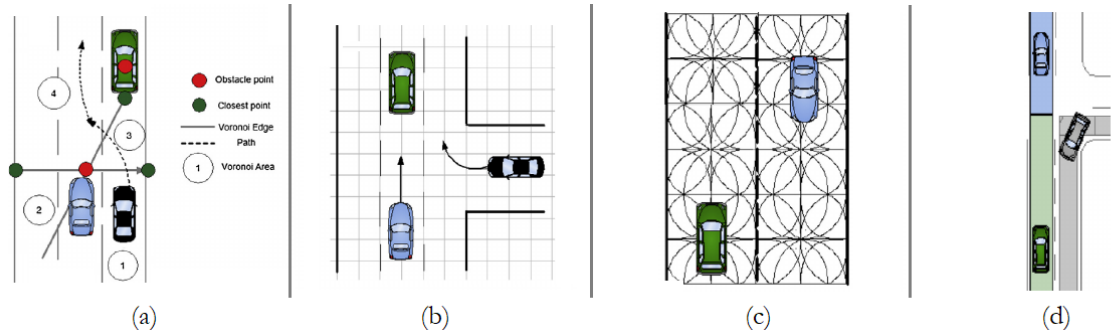


Figure 1.3: Search states for planning: (a) Voronoi Diagram [3], (b) Occupancy Grid [4], (c) State Lattice [5], and (d) Driving Corridor [6]

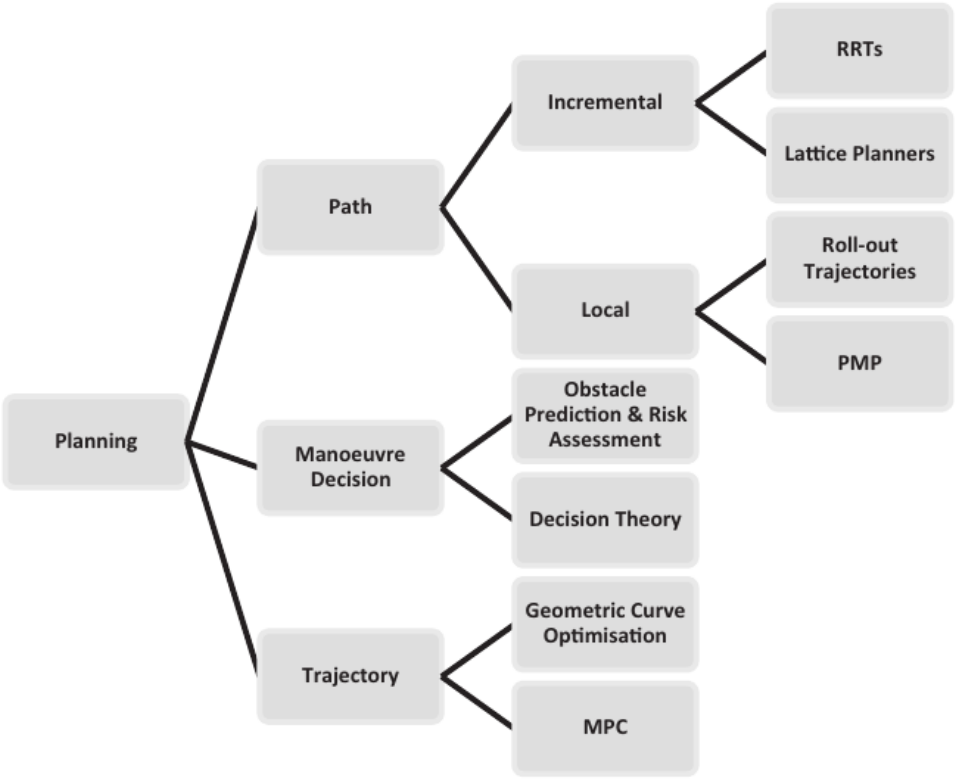


Figure 1.4: Classification tree for planning approaches. [2]

Path planning techniques Figure 1.4 gives an overview on the most common path planning techniques, with the classification between incremental and local search. While incremental search is concerned by finding the best sequence of actions through incremental sampling or discrete geometric structures, the local search goal is to find the best action from multiple final states [2].

Focusing on incremental search, a comparison between the two main search methods for path planning (Rapidly exploring Random Trees and Lattice Planners) is presented in [2] and illustrated here in Table 1.2.

RRTs	*Advantages*: – Kinematic and real-time feasibility – Quick search of free space – Advanced decision techniques for collision checking – Optimality guaranteed in newer implementations *Disadvantages*: – Jagged paths – Heavily dependent on the Nearest Neighbour heuristic – Collision checking for every expanded node – Advanced collision checking pre-suppose perfect environment knowledge
Lattice Planners	*Advantages*: – Low computational power needed – Guaranteed smoothness and optimality – Appropriate for dynamic environments – Compliance to vehicle's dynamics and kinematics *Disadvantages*: – Time inefficiency for evasive manoeuvers – May lead to exhaustive sampling or oscillations – Transferability

Table 1.2: Comparison of search space for planning. [2]

Manoeuvre planning techniques The manoeuvre planner has to compute the best and safest manoeuvre to undertake starting from the path provided by the path planner (given as a geometric sequence of waypoints to follow). The planner incorporates techniques which goal is to anticipate and understand the surrounding traffic situation. Such techniques are different from path and trajectory planning techniques, because they work on a higher-level basis: instead acting as a brain which filters the path planner results, giving the approval to the one that is coherent with the detected traffic situation (to then pass it to the trajectory generator in order to make a feasible trajectory out of it). These techniques can be divided into two categories, as previously shown in Figure 1.4: those that emphasize obstacle prediction and motion modelling, and those based on the modelling of the traffic environment. A detailed analysis of advantages and drawbacks of the main manoeuvre planning techniques can be found in [2, p.430].

Trajectory planning techniques The trajectory planner generates a trajectory from the final path received after the joint action of path and manoeuvre planners. This trajectory has to satisfy the motion model and the state constraints, while guaranteeing comfort for the passenger and smoothness for the trip. In order to generate the trajectory, a geometric curve is firstly selected, such that the smooth motion through the waypoints in input is assured. Then, this initial trajectory is optimised by using a cost function, solving an optimization problem according to the dynamic model and/or obstacles in the resulting trajectory. But as shown in Figure 1.4, the geometric curve optimization approach is not the only. A different approach that

combines the control and planning modules is called Model Predictive Control (MPC). In MPC, starting from a dynamic model for the vehicle, controller inputs sampled, generating the optimal trajectory. A detailed analysis of advantages and drawbacks of the main trajectory planning techniques can be found in [2, p.432].

It is important to notice that the three approaches (path, manoeuvre and trajectory planning) are typically combined into general frameworks that provide a complete plan for the autonomous vehicle.

1.1.2 Problems of classical planning approaches

As seen in the previous section, there are numerous drawbacks in using classical path planners (like RRT). Mainly, the planner can not avoid producing a not human-like trajectory, that can be sometimes generated. Another main drawback is that the resulting path needs to be smoothed and processed by the manoeuvre planner before being used for the following step, and this increases the planning time and the complexity of the general framework. Other notable constraints and limitations not illustrated here and in the Table 1.1 and 1.2 can be found in [2, pp.430, 432, 433].

1.2 Deep motion planning

In order to solve the problems related to classic motion planning techniques, it is possible to use a Convolutional Neural Network (CNN). This CNN can successfully learn a mapping from images obtained by a front-facing camera to steering angle by observing human experts. So, a neural network can replace the classical path planner in the path planning step.

1.2.1 NVIDIA's DAVE-2

In NVIDIA's DAVE-2 project (2016) a CNN went beyond mere pattern recognition, learning the entire steering pipeline of an automobile [7]. The groundwork was a DARPA project of known as DAVE (DARPA Autonomous Vehicle, 2004) [8] in which a sub-scale radio controlled car drove through a junk-filled alley way, . trained on hours of human driving in similar, but not identical, environments. Even DAVE's performance was not sufficient to provide an alternative to the classic modular approaches, DAVE-2 legitimated this alternative approach, with impressive results.

The system, that is implemented with the CNN shown in Figure 1.5, learns the internal representations necessary for the planning in a fully automatic fashion. Hence, the great advantage of this approach is that the CNN can detect automatically road features with, as input, only the human steering angle as the training signal. No explicit training has to be performed, not even for road outlines detection (see Figure 1.6 and Figure 1.7).

Compared to the classic motion planning approach, the end-to-end CNN system optimizes all processing steps simultaneously. As reported in [7], this will eventually lead to better performance and smaller systems, because the internal components self-optimize to maximize overall system performance. In classical approaches the optimization is restricted to each module and the inter-modular criteria are human-selected. Such criteria are selected for ease of human interpretation, and this does not guarantee the optimization of the system and its maximum

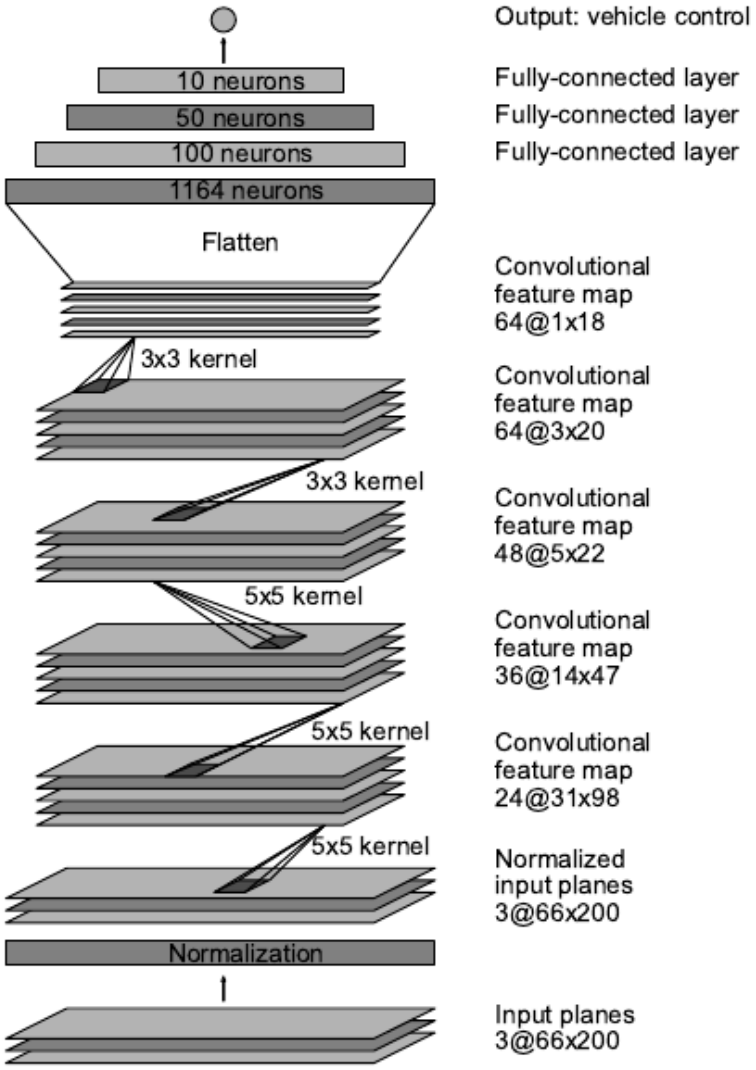


Figure 1.5: DAVE-2 CNN architecture. The network has about 27 million connections and 250'000 parameters. [7]

performance. Smaller CNN networks like the one implemented in DAVE-2 instead solve the problem with the minimal number of processing steps, and with a general optimization not reachable for a modular approach.

1.2.2 Hybrid motion planning

With the DAVE-2 system, the end-to-end learning approach got new power and has been applied by many others research teams, for example using LIDAR sensors for path generation [9]. The current project mutuate its design concept from [10], where an integrated approach has been proposed, presenting an architecture that includes the end-to-end learning project in a more classical framework. This approach will be called "hybrid approach".

In this work, a CNN has been trained on monocular image data (as in [7]), but the main change is the use of local history of image data. Thanks to the local history, it has been possible to obtain a CNN that implicitly represents the spatial environment around the vehicle, like

Figure 1.6: DAVE-2's CNN feature maps for an unpaved road image. Top: subset of the camera image sent to the CNN. Bottom left: Activation of the first layer feature maps. Bottom right: Activation of the second layer feature maps. This demonstrates that the CNN learned to detect useful road features on its own.[7]

Figure 1.7: DAVE-2's CNN feature maps for an image with no road. The activations of the first two feature maps appear to contain mostly noise, i. e., the CNN does not recognize any useful features in this image.[7]

parked cars or other obstacles commonly found in urban and residential areas. In fact, the drawback of using a single front-facing camera is that the provided field of view (FOV) is not enough in order to detect obstacles besides the vehicle. Those kind of obstacles are, in fact, not visible anymore when the part of the vehicle where the camera is positioned pass them, and this is a problematic aspect of using CNN. In fact, a CNN trained to drive on the right-hand side of the road tended to steer into parked cars because of this problem [10], and this does not guarantee safety. Moreover, the same problem has been noticed for sharp corners detection. All these problems, that arise while driving in residential areas, can be solved through the introduction of local history, calculated using the vehicles velocity data.

Probabilistic formulation for the planning problem From a very high level perspective, the planning problem can be reformulated as a probabilistic problem [10]:

- Given m as some input, like sensor information, map data or any localization information available.
- Given θ as some given background knowledge, like physics laws, traffic rules, perception principles...
- Given ω as the driving decision
- Given the following probabilities distributions:
  $m \sim P(m)$ probability of having a certain input
  $\theta \sim P(\theta)$ probability of having a certain background knowledge

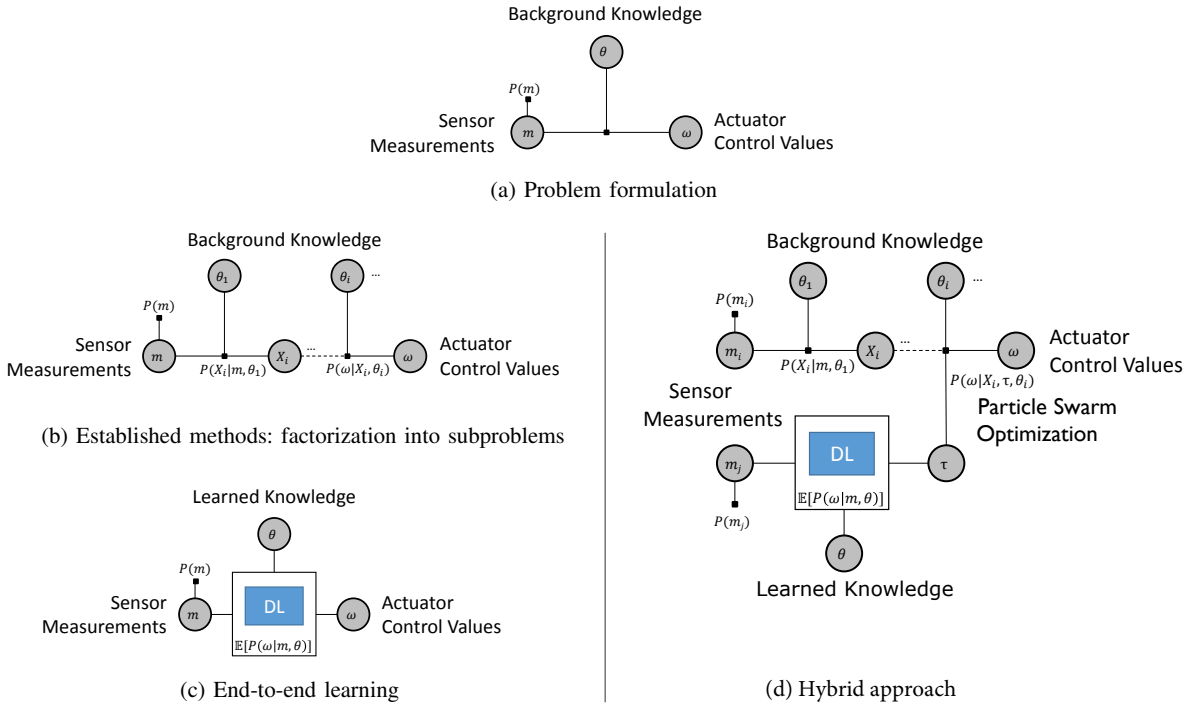
$\Rightarrow$ The probability of having a certain driving decision (influenced by the input and by the background knowledge) is then expressed as:

$$\omega \sim P\left(\omega | (m, \theta)\right) \tag{1.1}$$

A scheme of this general formulation is shown in Figure 1.8a. Since this general problem is too complex to be modeled and solved in its entirety, two approaches (that has been already seen before in a different formulation) are normally used:

- Classical or "mediated perception" approach (Figure 1.8b):
  Reduce the complexity of the problem factorizing it into modular subproblems, using intermediate representations like sensor features, object lists, interpreted scene, planned trajectory... Commonly seen subproblems are object detection, scene understanding or trajectory planning (as seen in Section 1.1).
- End-to-end or "behavior reflex" approach (Figure 1.8c):
  Using deep learning and big computation capabilities it is possible to solve the problem with implicit representations, as seen in Section 1.2.1 with the DAVE-2 system.

The hybrid approach proposed in [10] combines the previous two approaches (schematic shown in Figure 1.8d). Here, the trajectory output of the CNN is used as an input proposal for a probabilistic planning algorithm that optimize it. The optimization technique chosen in [10] is Kennedy & Eberhard's PSO algorithm (Particle Swarm Optimization [11] [12]), where each solution of the problem is represented as a particle of a swarm-like set, moving withing the solution space according to a set of rules and memorizing each time the best solution encountered.



(a) Problem formulation

(b) Established methods: factorization into subproblems

(c) End-to-end learning

(d) Hybrid approach

Figure 1.8: Factor graph representation of the theoretical problem formulation (a) that can be solved using a factorization into subproblems (b) or by an end-to-end learning approach (c). The hybrid approach (d) is a combination of deep learning and optimization incorporating safety criteria. [10]

The optimization performed by the planner has been proved useful in [10], reducing greatly the rejection rate of sampled trajectories due to obstacle collision, in particular for scenarios containing a lot of obstacles like parked cars. The vehicle controlled by the hybrid framework was able to learn to avoid semi-dynamic obstacles like parked cars and thus be learned in terms of spatial extension. Dynamic objects, on the other hand, still has been mainly ignored by the trained model, but the PSO planner allows to add dynamic objects information born by sensors and react improving the safety of the final trajectory.

1.2.3 The vanishing gradient problem

Deep neural networks are more difficult to train with respect to shallow neural networks. Hence, is not always true that having networks that learn better is a matter of stacking more layers. The main obstacle is problem of vanishing (or "exploding") gradients [13][14]. In fact, when deeper networks start converging, it has been noted a degradation problem: accuracy gets easily saturated and then degrades as rapidly as deep is the network, and this degradation is not caused by overfitting. A typical example reported in [15] is shown in Figure 1.9.

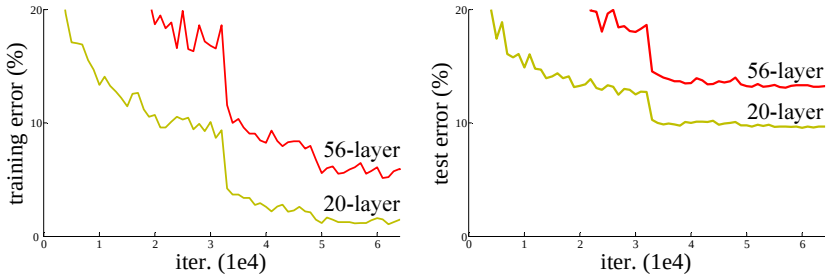


Figure 1.9: Vanishing gradient problem: training error (left) and test error (right) on CIFAR-10 dataset [16] with 20-layer and 56-layer "plain" networks.[15]

In [15] a solution to this problem has been proposed introducing the deep residual learning framework. The main idea behind the framework is to explicitly obtain a residual mapping for each block, in order to fit it in a easier way. If the normal mapping to fit in a network is, let's say, $\mathcal{M}(x)$ (where x is the layer's input), the residual mapping is defined as:

$$\mathcal{F}(x) = \mathcal{M}(x) - x \tag{1.2}$$

A mapping built in such way it has been proven easier to optimize [15]. The residual block that uses such residual mapping can be implemented with feedforward neural networks that use a "shortcut connection" in order to perform an identity mapping, adding the input of the block x to the outputs of the stacked layers (as shown in Figure **??**). The advantage of using identity shortcuts is that they do not add any extra parameter nor increase the computational complexity of the process.

A deep residual network (ResNet) has been proven easier to optimize than a "plain" net that simply stack layers. In fact, plain nets exhibit higher training error if their depth increase. Moreover, ResNets gain accuracy from increased depth avoiding the vanishing gradient problem, producing better outcomes (see the results reported in Figure 1.11).

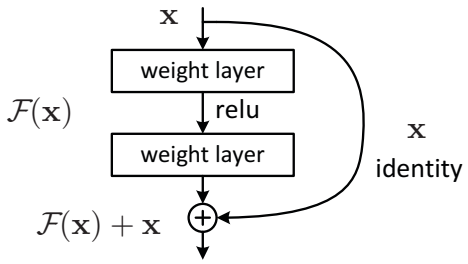


Figure 1.10: Residual block scheme. The shortcut connection is the one skipping the layers. [15]

Residual blocks can be built in different design fashions. In [17] different designs with different batch normalization (BN) layers and ReLU activation layers has been compared (Figure 1.12), and the residual unit shown in Figure 1.12e has been presented as easier to train while improving generalization along the network.

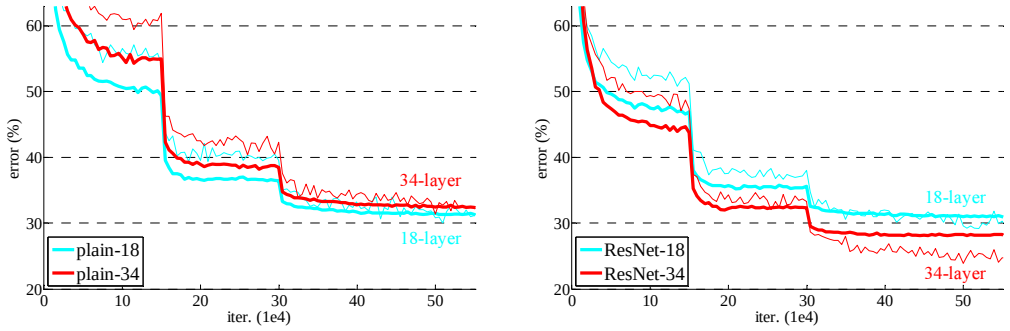
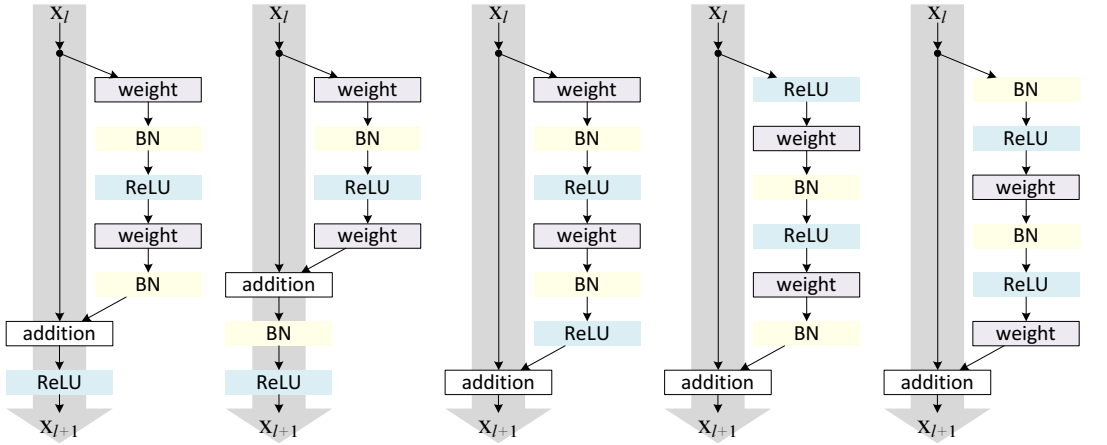


Figure 1.11: Residual networks training on ImageNet database. Thin curves denote training error, and bold curves denote validation error of the center crops. On the left the result obtained with two plain networks of 18 and 34 layers. On the right the result obtained with two ResNets of 18 and 34 layers. [15]



case	Fig.	ResNet-110	ResNet-164
original Residual Unit [1]	(a)	6.61	5.93
BN after addition	(b)	8.17	6.50
ReLU before addition	(c)	7.84	6.14
ReLU-only pre-activation	(d)	6.71	5.91
full pre-activation	(e)	**6.37**	**5.46**

Figure 1.12: Various usages of activation in residual blocks. All these units differs only for the order of the components. In the table are shown the classification errors (in percentages) for the CIFAR-10 test set. [17]

1.3 Project goal

This project propose a hybrid approach in order to solve the motion planning problem. As seen in the previous sections, a CNN can be used instead of a classical path planner in the framework. In this project, such CNN is trained to produce a sequence of steering angles as proposed in [10], but in this case producing 5 steering angles instead of 25, and with 2 meters between two adjacent steering angles instead of 1 meter. Then, this steering sequence is mapped into a non-parameterized path and, once the path is output, the trajectory planning is invoked to time-stamp the path.

DroNet for classification As reported in Section 1.2.3, residual networks are the best choice in order to solve the stated problem. The network used in the project is based on the so-called DroNet proposed in [18], that is basically a ResNet-8 architecture followed by a dropout layer of 0.5 and a ReLU activation layer. As shown in Figure 1.13, after the last ReLU layer the network is followed by two different fully-connected (fc) layers. The first one outputs the steering angle, and the second one a collision probability. Steering prediction in [18] has been implemented as a regression problem, while the collision prediction is addressed as a binary classification problem.

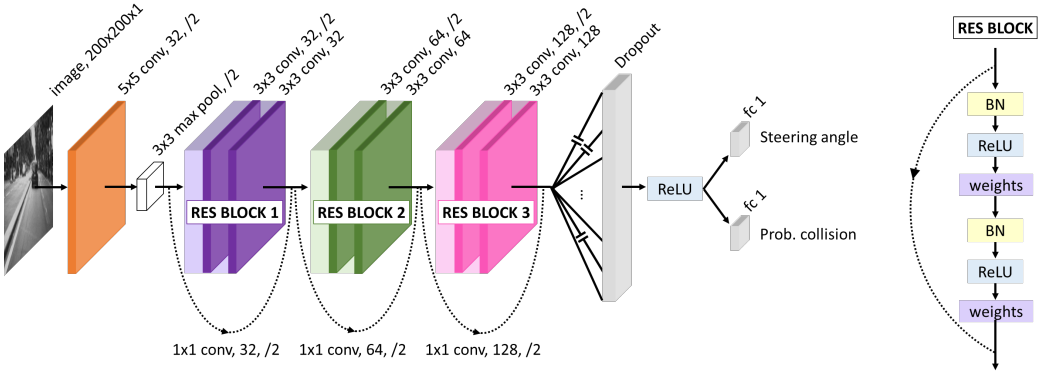


Figure 1.13: DroNet residual network scheme. [18]

The main difference between the DroNet approach and the one here presented is the steering angle prediction problem statement. In fact, as proposed in [19], this problem can be transformed from a regression problem of continuous values to a classification problem in which the steering angle is discretized, assigning angles to classes of 0.01 radians span. This is justified because the jitter for a human driver is measured to be within an error of 0.01 radians [19].

The network designed will be integrated into a bigger framework that will operate the optimization of the proposed trajectory. It is notable how the classification tree will change with this modification (Figure 1.14) with respect to the classical planning approach classification tree (Figure 1.4).

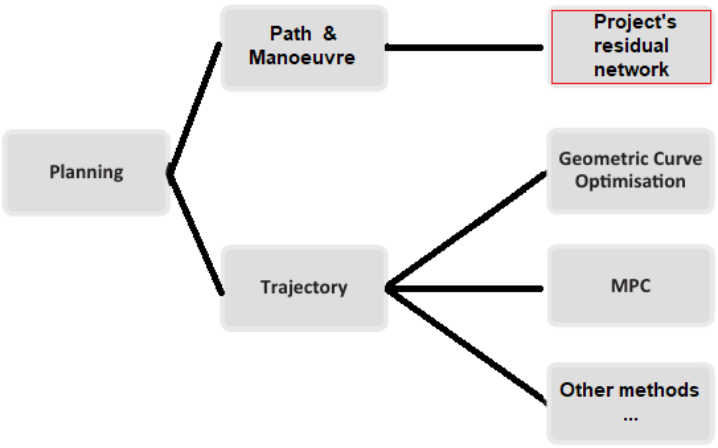


Figure 1.14: Revised classification tree for planning approaches with the inclusion of the network designed.

Hardware specifications For the implementation and analysis it has been used Python 3.6 with TensorFlow 1.13, running on Google colab and a DELL Latitude 5490. Google colab uses a Nvidia Tesla K80 with a Intel Xeon CPU (2.30GHz) and 12GB of RAM. The DELL Latitude 5490 uses a Nvidia GeForce MX130 with an Intel Core i7-8650U (1.90GHz) and 16GB of RAM.

Chapter 2

Learning Steering Angles

As discussed in the introduction, a single steering angle can be learned in an end-to-end fashion using a CNN. In this chapter will be presented the modified ResNet architecture and the dataset used to train it. The network will be able to predict only one angle given in input a single image.

2.1 Network Architecture

Each block of the network is comprised of three convolutional layers: two on the main path and one on the shortcut (Figure 2.1). The network's body parameters are summarized in Table 2.1. The output shape of a Conv2D layer is calculated by

$$SO = \text{ceil}\left(\frac{SI - K + 2P}{S} + 1\right) \tag{2.1}$$

with SO, SI are either height or width of the output and input, respectively. K is the size of the kernel, P is the number of rows or columns padded into the input, S is the stride. Using Equation 2.1 and Table 2.1, it is straightforward to verify that the size of output of Conv2D_c matches with the output of Conv2D_b for every ResNet block.

The classifier part of the network, as shown in Figure 2.1, is made of two dense layers. The first activation is ReLU. The second is activated by Softmax to output a one-hot representation of a single steering angle. A classifier's parameters is summarized in Table 2.2

The underlying reason of the separation between the CNN part and the Dense part is that,

Stage	Layer	Number of kernels	Kernel size	Stride	Padding
1	Conv2D_a	32	3	2	same
1	Conv2D_b	32	3	1	same
1	Conv2D_c	32	1	2	same
2	Conv2D_a	64	3	2	same
2	Conv2D_b	64	3	1	same
2	Conv2D_c	64	1	2	same
3	Conv2D_a	128	3	2	same
3	Conv2D_b	128	3	1	same
3	Conv2D_c	128	1	2	same

Table 2.1: Modified ResNet CNN body parameters

Layer	Parameters
Dense_1	800
Dropout	0.5
Dense_2	number of classes

Table 2.2: Modified ResNet classifier parameters

in this architecture, the ResNet-made body should learn better how to output useful feature maps which probably contains roads shape and drivable area, while the classifier on the top should learn how to output the steering angle given the provided feature map.

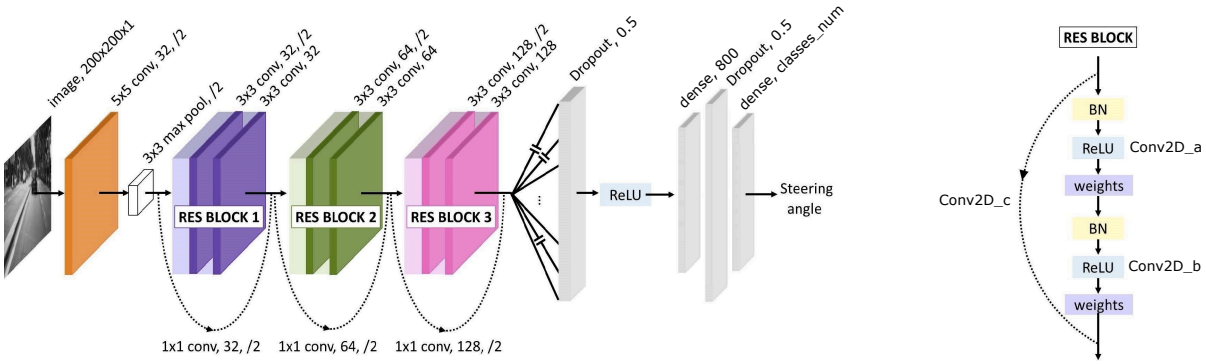


Figure 2.1: Modified ResNet architecture

2.2 Data Preparation

2.2.1 Definition of learning

The dataset used for training the network is Udacity dataset Challenge 2.[1] This dataset contains 6 ROS-bag files, each describes a 15 to 45 minutes trajectory on suburban roads in different lighting condition. These bag files is preprocessed with *udacity-dataset-reader*[2] to create a *csv* file containing the following information:

- Camera position (left, center, or right)
- File name
- Steering angles in radians
- Longitude
- Latitude
- Speed

If the steering angle prediction problem is modeled as a regression problem like [7; 10; 18], the generated *csv* file can be used immediately. However, as mentioned in Chapter 1, this regression approach has the major disadvantage of the reliability of the predicted angle. The proposal of

[1]Source: www.github.com/udacity/self-driving-car/tree/master/datasets/CH2
[2]Source: www.github.com/rwightman/udacity-driving-reader

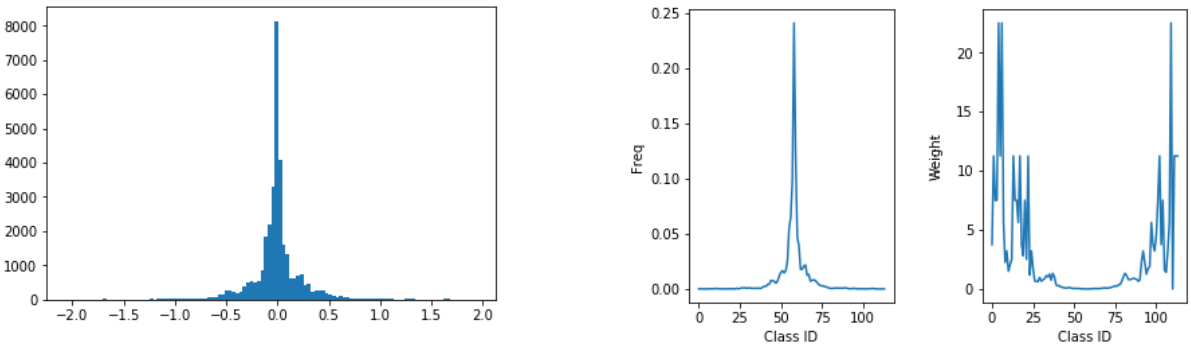


Figure 2.2: Classes' frequency (left) and classes' weight (right) distributions

[19], followed in this project, is to change the regression problem into a classification problem. In this new formulation the network learns to predict the class of a steering angle, rather than the angle's value. Here, a class represents an interval of the whole steering angle spectrum.

The most prominent advantage of learning a classification problem is that the network performance can be measured by the accuracy metric, in addition to the cross entropy loss. The accuracy conveys more information about model's prediction than the root mean square error which is the only indicator in the regression problem. The trade-off is a larger network since the output layer has as many neurons as the number of angle classes, compared to one single neuron in regression problem. In addition, the discretization of steering angle spectrum leads to one more hyperparameter which is the width of each class. Because of the trade-off above is minor compared to the aforementioned advantage, the learning of steering angle in this project is formalized as a classification.

2.2.2 Dataset treatment

To enable the network to learn a classifier, the steering angle spectrum is discretized by a step of 2 degrees. The histogram of steering angle classes is shown in Figure 2.2. From this histogram it is possible to notice that the dataset is dominated by classes associated with neutral position of steering wheel (angles near zero). This imbalance needs to be resolved to keep the network from biasing toward neutral value. Inspired by the median frequency balancing method in [20], each class is assigned a weight calculated by:

$$w = \frac{\text{median frequency}}{\text{frequency of this class}} \tag{2.2}$$

Here frequency a class is the ratio of the number of sample of this class over the number of sample in the whole dataset. The median frequency is the median of the all classes' frequency. The distribution of classes' frequency and classes' weight are shown in Figure 2.2. These weights are used to scale the error during back-propagation so that less likely ranges of angles have greater influence while more likely values have less influence.

2.3 Model Training

The model's weights are initialized randomly and it is trained by minimizing the cross entropy loss function with Adam optimizer. The choice of hyperparameters is shown in Table.2.3

Hyperparameter	Value
Batch size	200
Learning rate (α)	0.001
First momentum (β_1)	0.9
Second momentum (β_2)	0.999
Learning rate decay	0.0

Table 2.3: Hyperparameters' value

The evolution of model's loss and accuracy are respectively shown in Figure 2.3 and Figure 2.4. As can be seen from these graphs, in the first epochs the loss decrease rapidly, and the accuracy increase with a slower pace. After that the rate of improvement of both loss and accuracy gradually reduced. The validation accuracy of every classifier fluctuates at more than 67%.

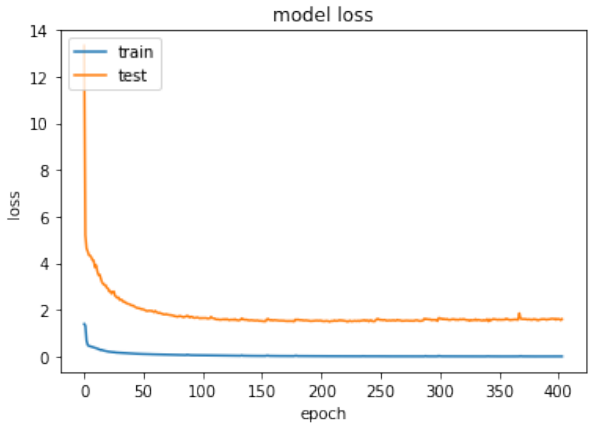


Figure 2.3: Evolution of loss with respect to the number of epochs on the training set and the unseen validation set. In these figure the horizontal axis corresponds to the number of epoch while the vertical axis denotes the value of loss.

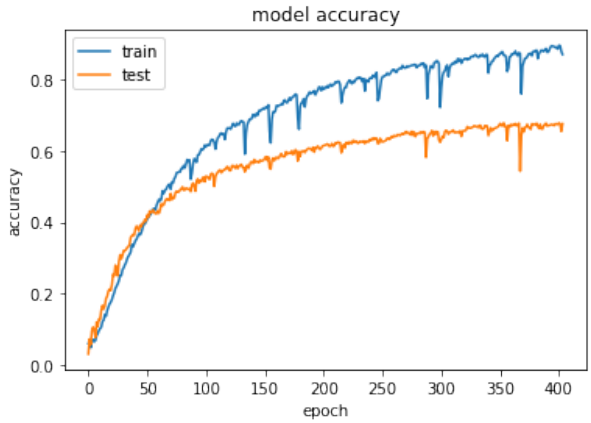


Figure 2.4: Evolution of accuracy of with respect to the number of epochs on the training set and the unseen validation set. In these figure the horizontal axis corresponds to the number of epoch while the vertical axis denotes the ratio of accurately predicted samples.

2.4 Model Performance

2.4.1 Qualitative performance

The comparison between the histograms of predicted angle classes and their ground truth on validation set are shown in Figure 2.5. This figure indicates a relative match between the predicted distribution and the true distribution.

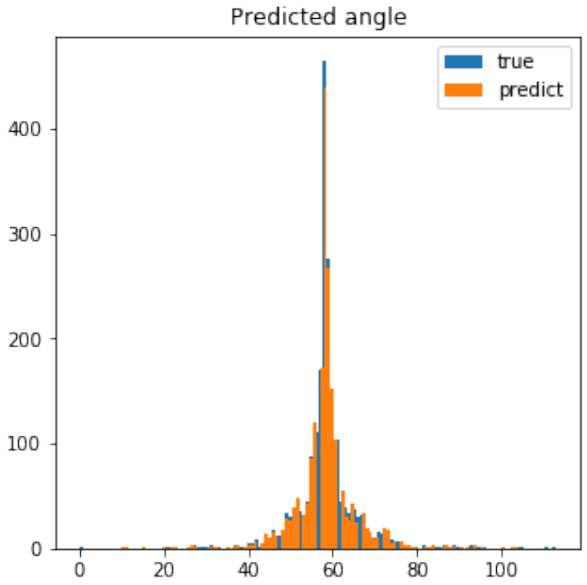


Figure 2.5: Predicted angle classes distribution compared to the ground truth

2.4.2 Quantitative performance

Using the same approach as [18], the quantitative performance is measured by two metrics which are the Root Mean Square Error (RMSE) and the Explained Variance score. The performance over these metrics of the classifier in the path planning model compared to some other architects is shown in Table.2.4

Model	RMSE	EVA
Random baseline	0.3 ± 0.001	-1.0 ± 0.022
Constant baseline	0.2129	0
DroNet	0.109	0.737
Project's ResNet	0.1383	0.7246

Table 2.4: Path planning model quantitative performance

Chapter 3

Learning a Geometrical Path

As discussed in the previous chapter, a single steering angle can be learned in an end-to-end fashion. Nevertheless, knowing the steering angle is just half of the task of driving an autonomous vehicle. The other control signal need to be provided is the vehicle's speed. There are no means of inferring a vehicle speed given a single steering angle at the same time instance. However, classical trajectory planning literature suggests that a geometrical path can be timestamped to generate a velocity profile. Therefore, this chapter will modify the ResNet-8 architecture as well as the data set to enable the network to learn a geometrical path end-to-end.

3.1 Network Architecture

Inspired by [10], a path can be implicitly encoded as a sequence of steering angles, each of which is applied to a predefined traveling distance. Based on this insight, a network can learn a path by learning a sequence of steering angles. This leads to the replacement of the last layer of the architecture in Chapter.2 by an array of classifiers. Like the single classifier of the previous chapter's network, each classifier is made of two dense layers: the first activation is ReLU and the second is activated by Softmax to output a one-hot representation of a single steering angle. A classifier's parameters is summarized in Table 3.1

Putting the body and the array of classifiers together to get the complete architecture as shown in Figure 3.1 and Figure 3.2.

The motivation of this architecture is that the ResNet-made body will learn to output an useful feature map which probably contains roads shape and drivable area, while the classifiers on the top will learn to output the steering angle given the provided feature map.

Layer	Parameters
Dense_1	800
Dropout	0.5
Dense_2	number of classes

Table 3.1: Classifier parameters

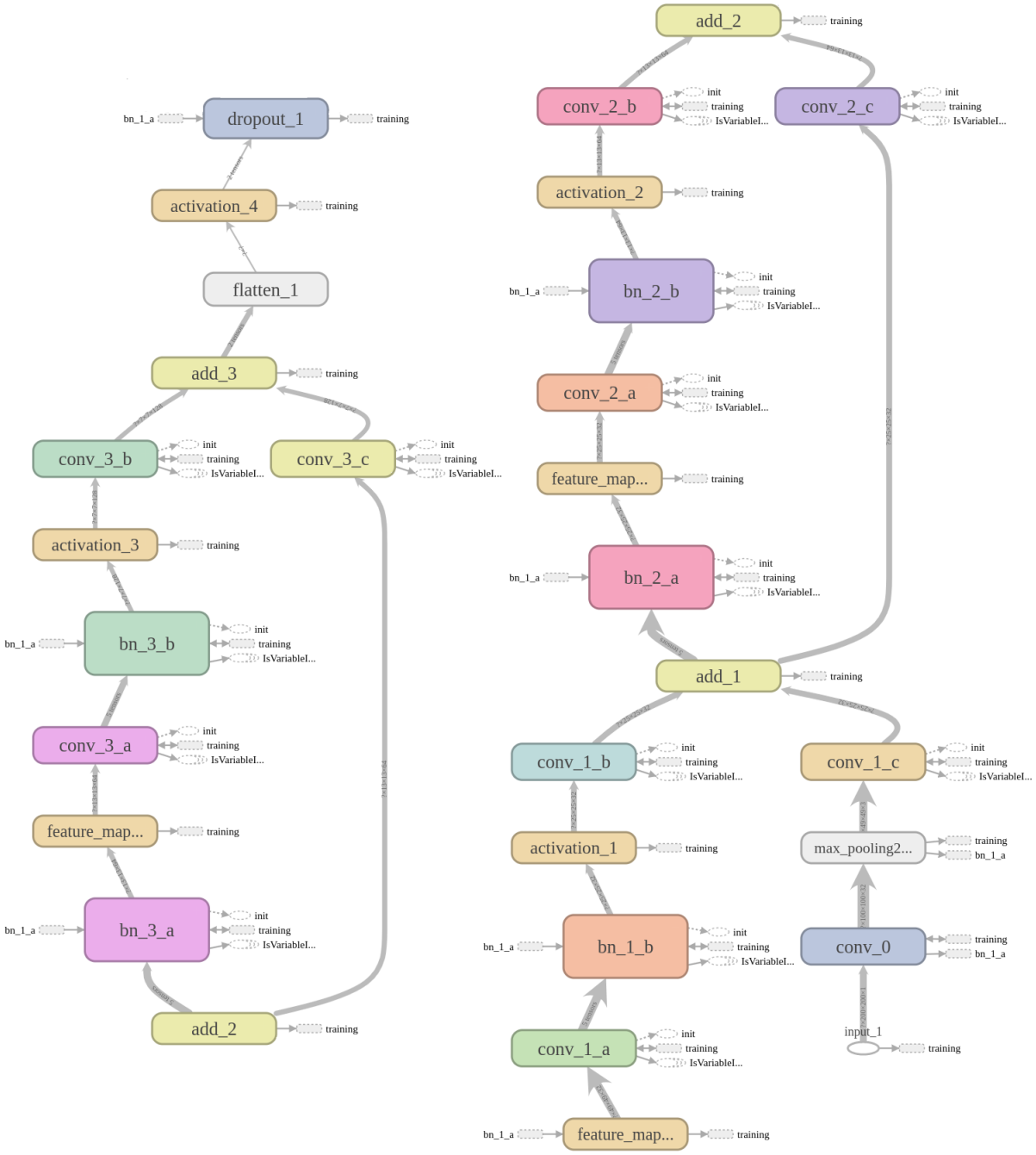


Figure 3.1: Path planning architecture (CNN part)

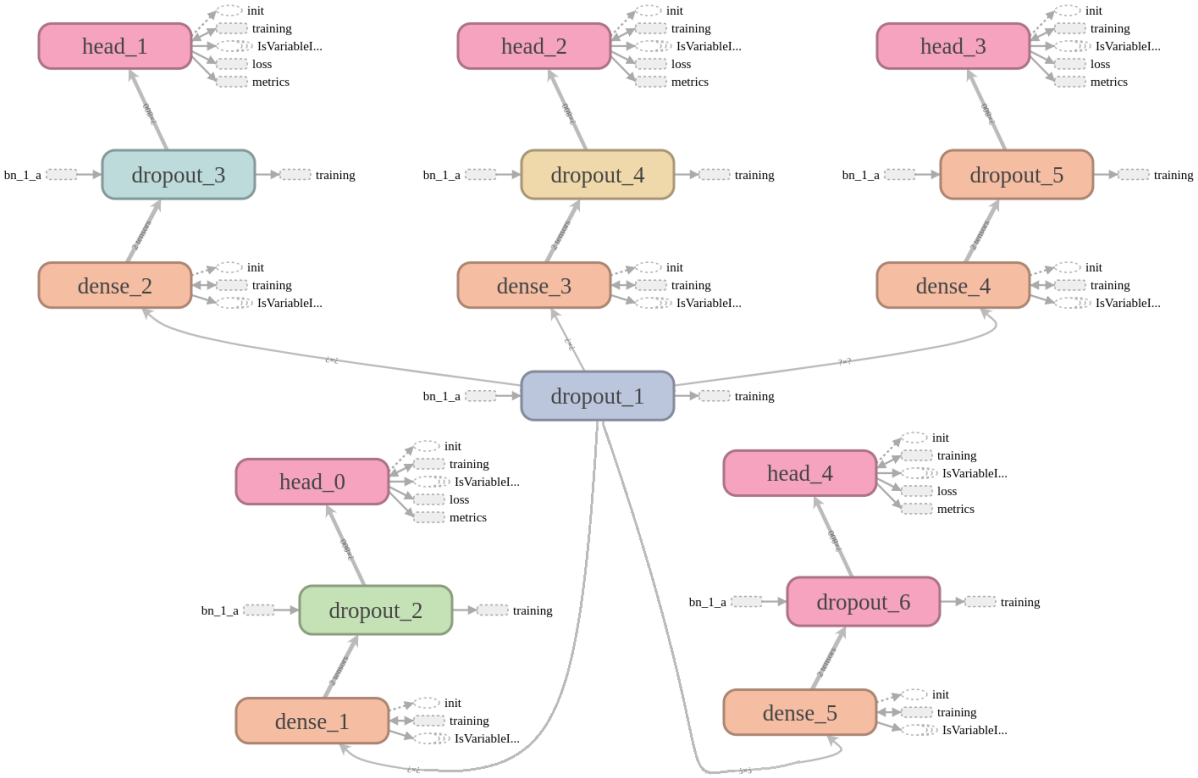


Figure 3.2: Path planning architecture (fully connected part)

3.2 Dataset Preparation

Given a single image, the target of the path planning network is to output a sequence of 5 steering angles, each of which is applied to 2 meters of traveling distance. As a result, a training sample is comprised of

- X: a 3D array represents an image
- y: a list of one-hot vectors. This first vector denotes the class of the steering angle associates with the frame represented by X. The i-th vector represents the steering angle of the frame $i \times 2$meters away from X.

Such definition of y suggests that the data set used to train path planning model needs to explicitly contains the distance information between two adjacent labels. This distance can be retrieved from the longitude and latitude of each frame in the original Udacity dataset.

With the definition of longitude and latitude in Figure 3.3, an Earth-centric Cartesian coordinate can be defined as following:

- Origin is placed at Earth's center
- x-axis goes from the Origin to the between intersection 0-latitude and 0-longitude circle
- z-axis goes from the Origin to the North pole
- y-axis is the cross product between z-axis and x-axis

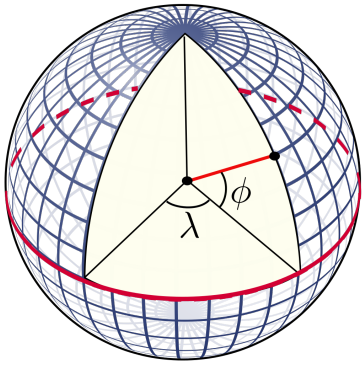


Figure 3.3: Longitude (λ) and latitude (ϕ) definition (source: Wikipedia)

The coordinate in this Cartesian frame is calculated based on longitude and latitude by the following equation:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} R\cos(\phi)\cos(\lambda) \\ R\cos(\phi)\sin(\lambda) \\ R\sin(\phi) \end{bmatrix} \tag{3.1}$$

Using Equation 3.1, the frame-to-frame distance is calculated as the distance between two points in 3D space. An example of training sample's y is shown in Figure 3.4

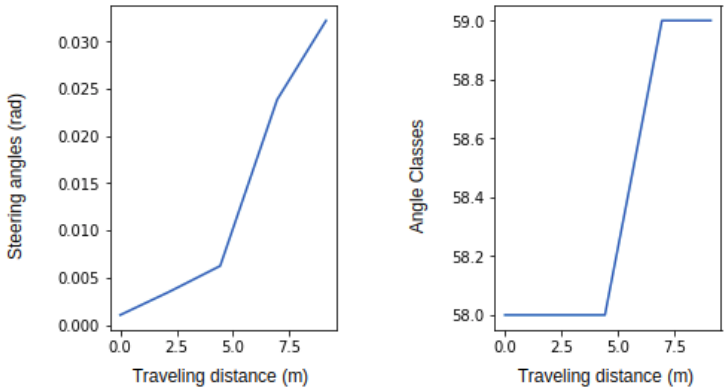


Figure 3.4: A training sample's labels (y)

The training data is generated by the following procedure:

1. Import the *interpolated.csv* which contains the original Udacity dataset
2. Initialize the training data to be an empty dictionary
3. If length of training data less than 60% of the length of *interpolated.csv* file go to step 4, otherwise finish.
4. Randomly choose a sample in *interpolated.csv*
5. Assign the image associated with this sample to the key X of the dictionary
6. Going forward from this sample, find another four frames, each of which is at least 2 meters away from its predecessor.

7. Assign the class of steering angles of the frame associated with X and those four frames to key y of the dictionary
8. Go to step 3

Upon completely being generated, the training data is divided into training set and validation set with the ratio of 5 to 1.

3.3 Training

The model's weights are initialized randomly and it is trained by minimizing the cross entropy loss function with Adam optimizer. The choice of hyperparameters is the same of the one shown in Table 2.3. The evolution of model's loss and accuracy are respectively shown in Figure 3.5 and Figure 3.6. As can be seen from these graphs, in the first five epochs the loss decrease rapidly, and the accuracy increase nearly with the same pace. After that the rate of improvement of both loss and accuracy gradually reduced. While the training loss keeps decreasing as the training process continues, the validation loss cannot be reduce further and starts to increase after 25 epochs. Since the validation loss can not longer be reduced after 25 epochs, the validation accuracy of every classifier fluctuates at more than 73%.

3.4 Model Performance

3.4.1 Qualitative performance

The comparison between the histograms of predicted angle classes and their ground truth on validation set are shown in Figure 3.7. This figure indicates a relative match between the predicted distribution and the true distribution.

In addition, the normalized confusion matrix of the first classifier is shown in Figure 3.8. This matrix features a clear, large magnitude main diagonal. This means the majority of predicted angle classes is actually the true class. Nevertheless, there are a few strong cells in the bottom Figure 3.8 implying that the classifier fail to the predict the class of extreme right angles.

3.4.2 Quantitative performance

Using the same approach as [18], the quantitative performance is measured by two metrics which are the Root Mean Square Error (RMSE) and the Explained Variance score. The performance over these metrics of each of five classifiers in the path planning model compared to some other architects is shown in Table.3.2

3.4.3 Visualize layer activation

To support the hypothesis formed in the first section of this chapter: the ResNet-made body will learn a useful feature map from the input images. The outputs of each ResNet block are displayed in Figure 3.9. This figure shows that the first block recognizes objects like lane mark and vehicles, while the second block recognize the drivable area. The last block learns a down sample mapping.

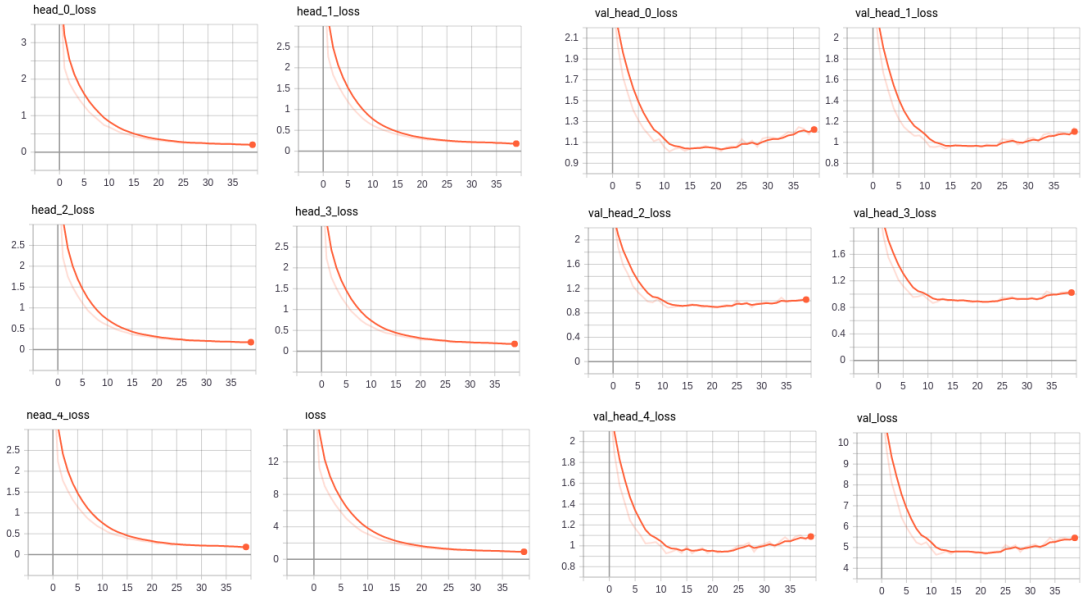


Figure 3.5: Evolution of loss of each classifier and the whole model with respect to the number of epochs on the training set and the unseen validation set. In these figure the horizontal axis corresponds to the number of epoch while the vertical axis denotes the value of loss.

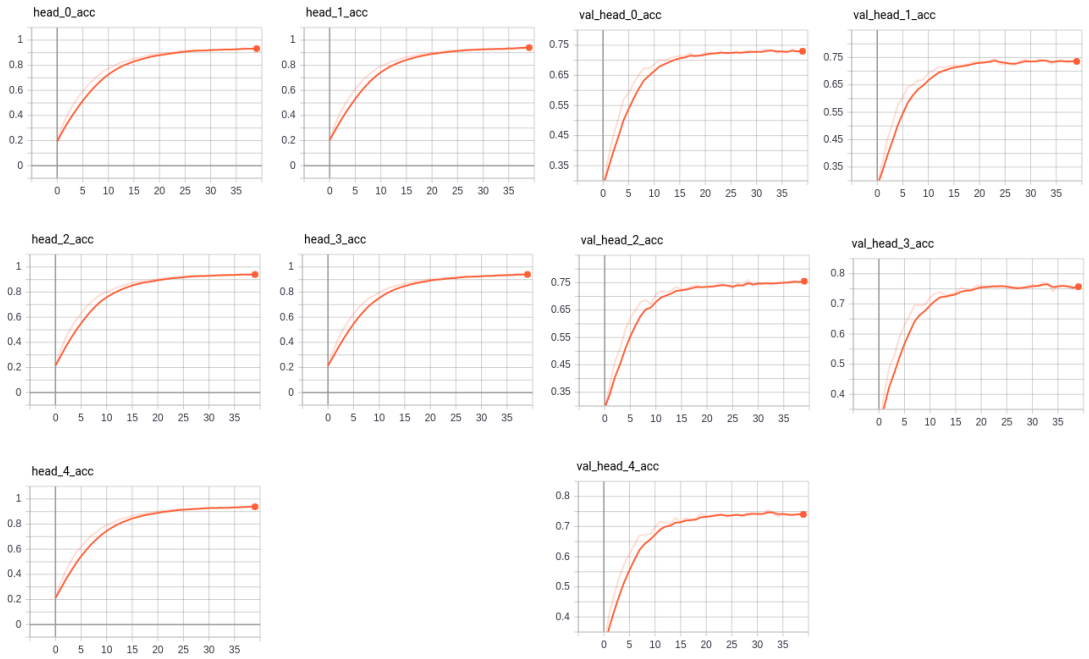


Figure 3.6: Evolution of accuracy of each classifier with respect to the number of epochs on the training set and the unseen validation set. In these figure the horizontal axis corresponds to the number of epoch while the vertical axis denotes the ratio of accurately predicted samples.

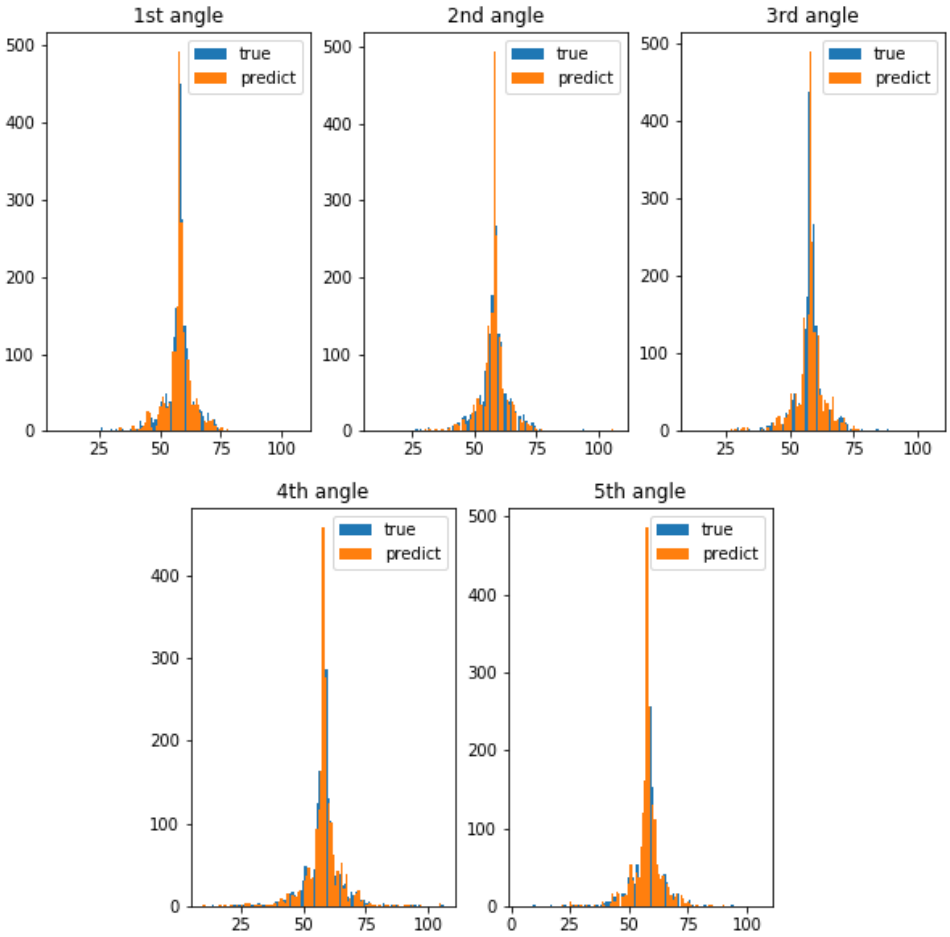


Figure 3.7: Predicted angle classes distribution of each classifier compared to their ground truth

Model	RMSE	EVA
Random baseline	0.3 ± 0.001	-1.0 ± 0.022
Constant baseline	0.2129	0
DroNet	0.109	0.737
Head_0	0.1101	0.8245
Head_1	0.1159	0.8042
Head_2	0.1012	0.8491
Head_3	0.1159	0.8020
Head_4	0.1050	0.8409

Table 3.2: Path planning model quantitative performance

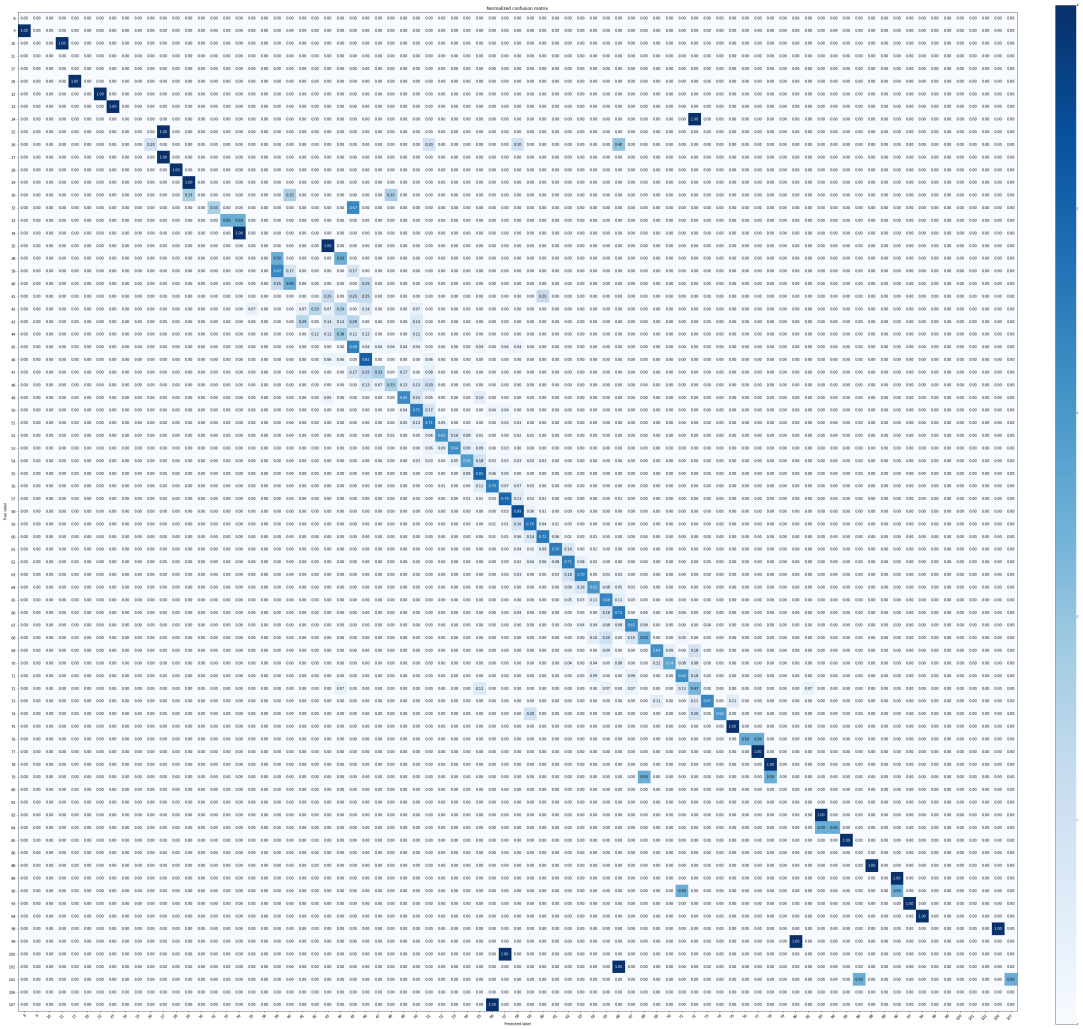


Figure 3.8: Normalized confusion matrix of the first classifier.

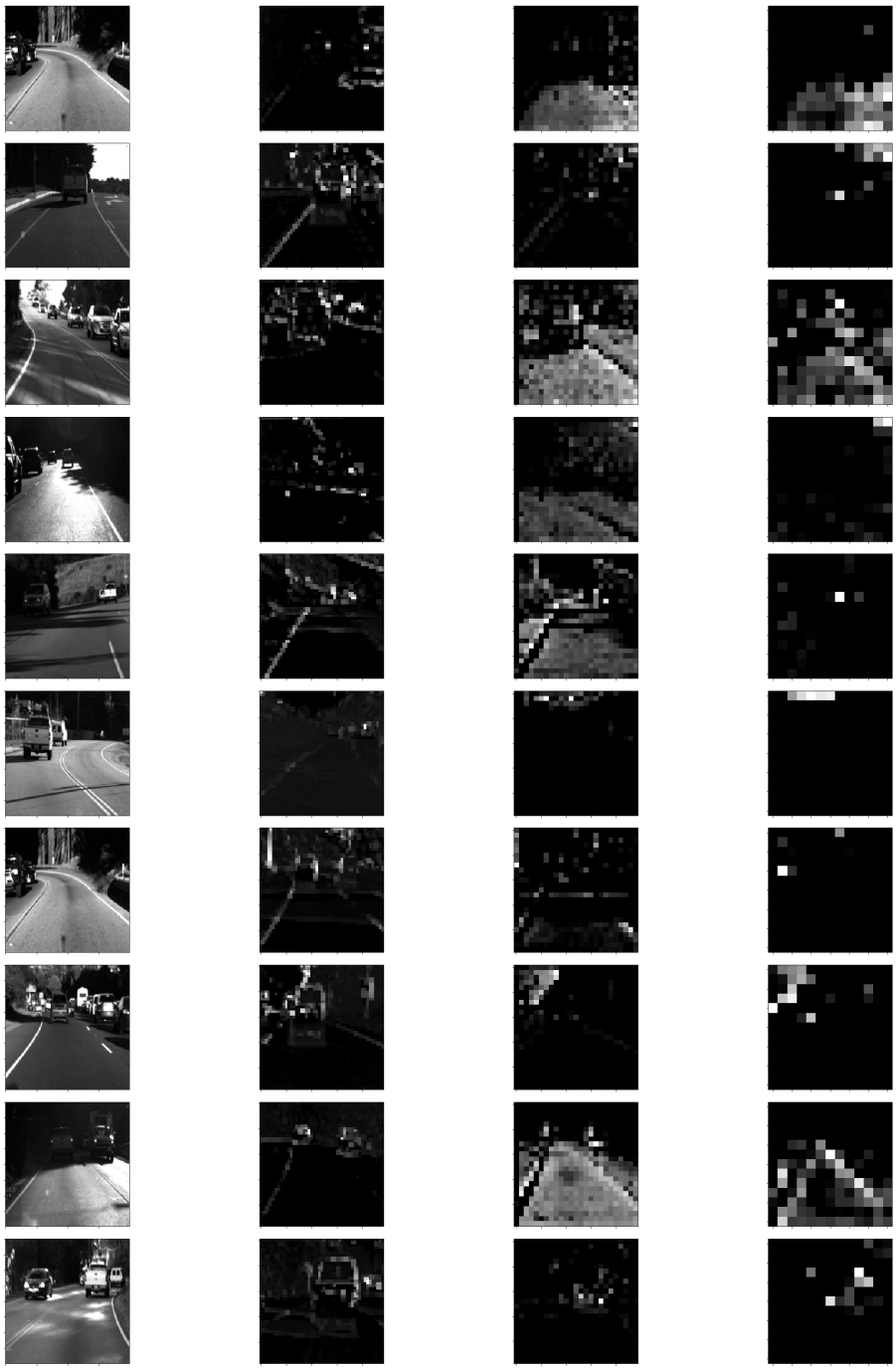


Figure 3.9: Output of each ResNet block with respect to different input images. From left to right, the images are respectively the input image, output of the first, second, and third block.

3.5 Interpret a steering sequence to a path

Since the motion of car-like vehicles is constraint to be circular around its Instantaneous Center of Rotation (ICR) (Figure 3.10), a sequence of steering angles can be interpreted to a geometrical path (i.e. sequence of waypoints) by applying each angle in the sequence to a predefined traveling distance s meters.

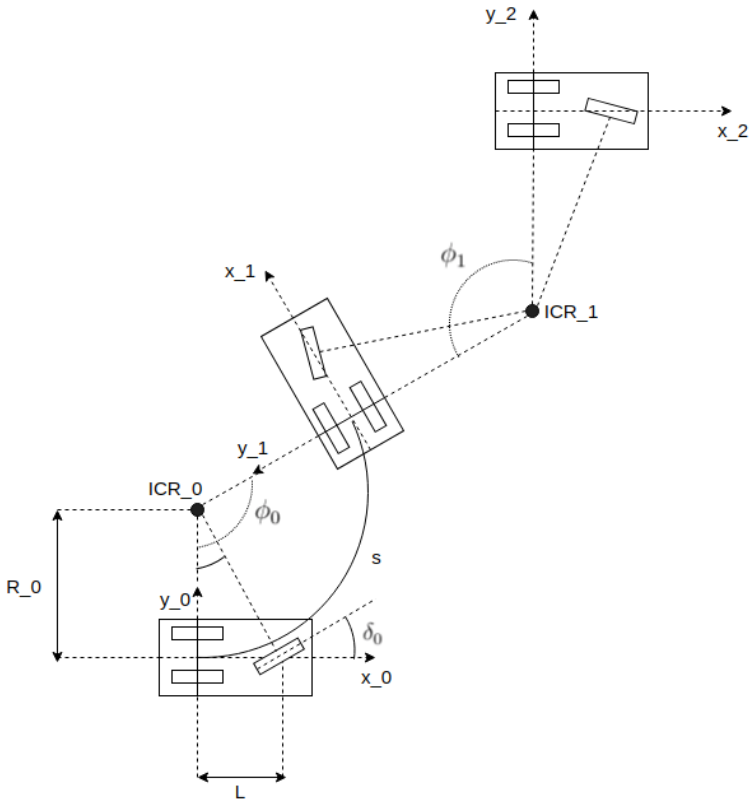


Figure 3.10: Car-like vehicle's motion diagram with two different value of steering angles. Each of these angles is applied to an arc distance of s meters.

In Figure 3.10, L is the distance between the front and rear axel. $\delta_i (i = 0, \bar{N})$ is a steering angle ($N + 1$ is the length of steering angles sequence). At each time instance, the pose of the vehicle is represented by the pose of the local frame attached to the center of its rear axel - $O_i x_i y_i z_i (i = 0, \bar{N})$. x_i goes from the rear axel to the front axel, and is perpendicular to these axels. z_i is orthogonal to the plane of motion, and pointing outward. y_i is defined such that $O_i x_i y_i z_i$ is right-handed. The target is to calculate the position of the center of the front axel relative to the local body frame at the presence - $O_0 x_0 y_0 z_0$.

Assume the vehicle's motion is planar, the transformation from frame $O_i x_i y_i z_i$ to frame $O_{i+1} x_{i+1} y_{i+1} z_{i+1}$ is described by

$$ {}^{i}T_{i+1} = \begin{bmatrix} Rot_{z,\phi_i} & {}^{i}t_{i+1} \\ 0_{1\times 2} & 1 \end{bmatrix} \tag{3.2} $$

Here, ${}^{i}t_{i+1}$ is the coordinate of O_{i+1} in frame i, and Rot_{z,ϕ_i} is the matrix represents the

rotation around z-axis by an angle ϕ_i.

$$Rot_{z,\phi_i} = \begin{bmatrix} \cos\phi_i & -\sin\phi_i \\ \sin\phi_i & \cos\phi_i \end{bmatrix} \tag{3.3}$$

The radius of the circular motion around ICR_i is

$$R_i = \frac{L}{\tan\delta_i} \tag{3.4}$$

Equation 3.4 implies that when the steering angle is close to zero (i.e. the steering wheel is kept at the neutral position), the radius of motion approach infinity, hence a straight motion. Given R_i, the coordinate of O_{i+1} in frame i is

$${}^{i+1}t_i = \begin{bmatrix} R\sin\phi_i \\ R\,(1-\cos\phi_i) \end{bmatrix} \tag{3.5}$$

In Equation 3.3 and Equation 3.5, ϕ_i is the angle between x_i and x_{i+1}. As can be seen in Figure 3.10, this angle is calculated by

$$\phi_i = \frac{s}{R_i} = \frac{s\tan\delta_i}{L} \tag{3.6}$$

With Equation 3.6 and Equation 3.5, the transformation from frame i to frame $i+1$ in Equation 3.2 is fully defined.

The local position of the center of the front axel in homogeneous form is

$${}^iL_i = \begin{bmatrix} L \\ 0 \\ 1 \end{bmatrix} \tag{3.7}$$

This position is transformed into the local frame at the presence by the following equation

$${}^0L_i = {}^0T_i{}^iL_i = \prod_{j=1}^{i} {}^{j-1}T_j{}^iL_i \tag{3.8}$$

Chapter 4

Learning a Geometrical Path with LSTM Layer

In the last chapter a modification is made to the output of layer of the original to ResNet-8 to enable the model learn a sequence of steering angles which is later interpreted into a geometrical path. This modified model possesses a relatively good performance in terms of quantitative metrics - RMSE and EVA. However, to obtain such performance, the model is made of a large number of parameters (25 millions). In addition, the inference of steering angle based on only one input image implies a pure geometric solution where the path is generated given a representation of drivable area learnt by the ResNet-made body. Since a sequence of steering angles implies a time order, it might be helpful if the network can learn a temporal relation among the steering angles. This chapter explores the application of recurrent layer to enable the model learning such time relation.

4.1 Network architect

Similar to the architect in the previous chapter, the model here is made of a ResNet-body and an array of classifiers. The difference is that to produce a temporal input to the model, three frames with 2-meter interframe distance is fed to model. Each of these frames is passed through an individual ResNet body to produce a feature map. Then, this feature map goes to an LSTM layer where it is fused with other feature maps. Finally, the last LSTM's hidden state is used as the initial state of the sampling architect which is inspired by DeepJazz.

It's worth to notice that the sampling architect is initially used in the context of NLP and time series processing. The application of this architect here is motivated by the belief that the underlying temporal relation among steering angles in a sequence is similar to that of text or time series signal.

At the training phase, the input to each LSTM cell in the sampling architect is the true label of the previous prediction. The training phase architect is shown in Figure 4.1. However, at the testing phase, such true labels are not available, instead the predicted label of a classifier is used as the input to its successor. As a result the architect in the testing phase shown in Figure 4.2 is different to the training phase.

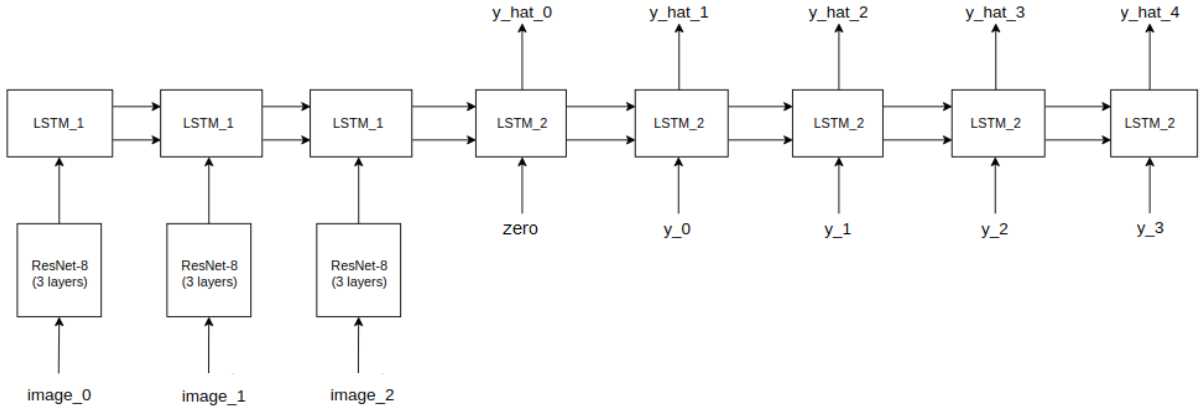


Figure 4.1: Path planning architect with LSTM layer in training phase

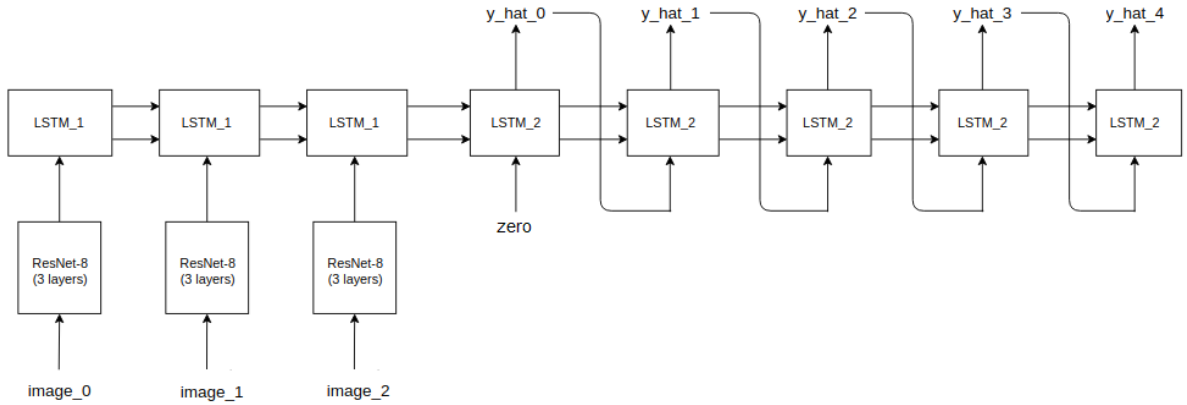


Figure 4.2: Path planning architect with LSTM layers in testing phase

4.2 Training

The choice of hyperparameters for training is kept the same as Chapter.3, except the batch size is decreased to 50 so that a batch can fit into our GPU's memory. In addition, the loss is modified by setting the weight of the first classifier's loss to 3 (the default value is 1)

The training process is shown in Figure 4.3 and Figure 4.4. Similarly to Chapter.3, the training loss decreases rapidly, then slows down, while the validation loss increases as the model begins to overfit the training set. After training for 45 epoches, the model with best weights achieves the accuracy of more than 70% for every classifier.

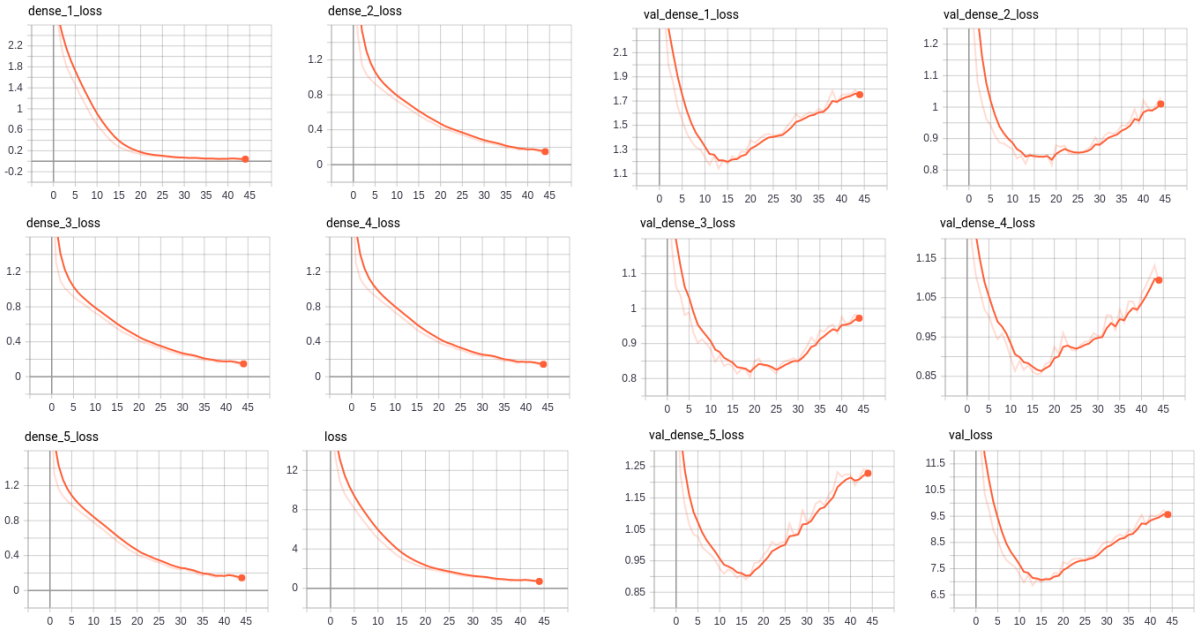


Figure 4.3: The evolution of training loss of the Path planning architect with LSTM layers.

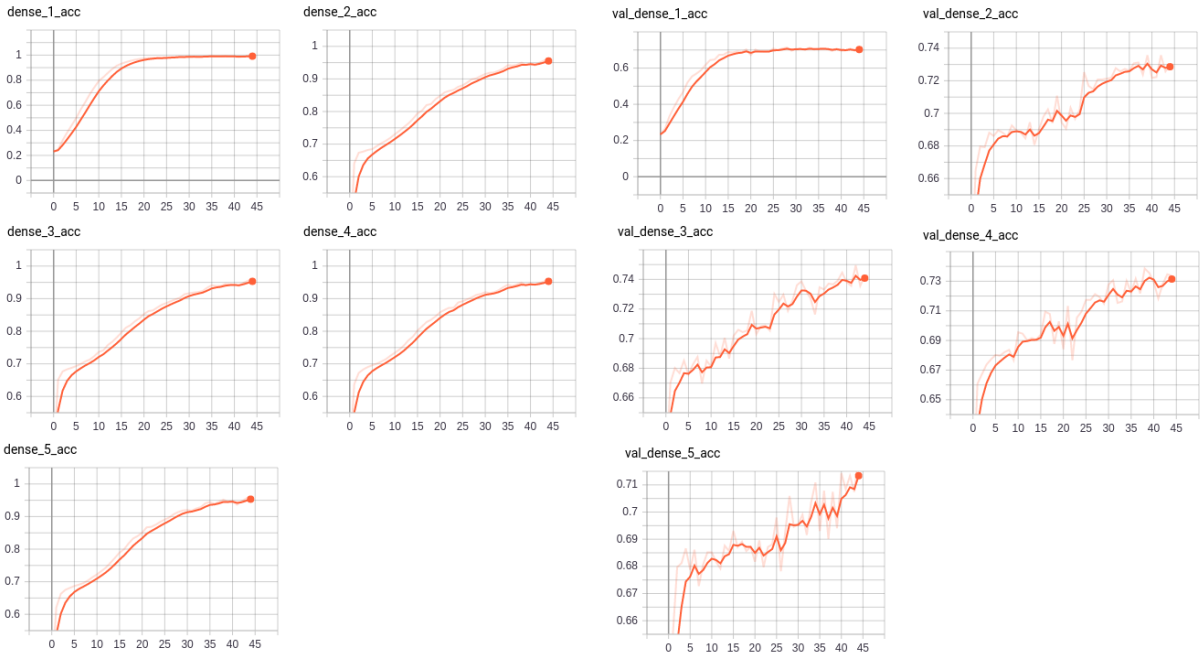


Figure 4.4: The evolution of accuracy of all classifiers in the Path planning architect with LSTM layers.

4.3 Model Performance

4.3.1 Qualitative performance

The distribution of each predicted angle is compared to its ground truth in Figure 4.5. It can be seen that all predicted distributions match relatively well with their ground truth.

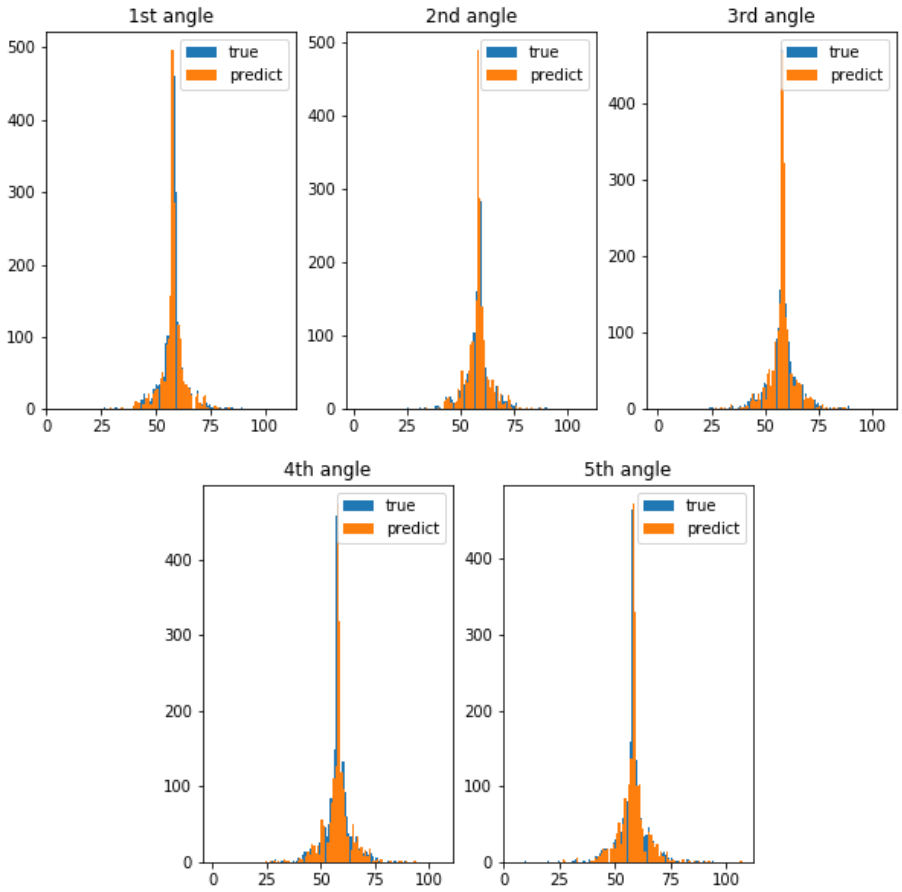


Figure 4.5: Distribution of predicted labels compared to their ground truth on validation dataset

4.3.2 Quantitative performance

The RMSE and EVA metric of path planning model with LSTM layer compared to the DroNet model [18] and the model in Chapter 3 are shown in Figure 4.6. Here is shown that the path planning model with LSTM layer is outperformed by the original version in Chapter 3 and the DroNet. This is due to the fact that the classifier accuracy in training phase is just 70%. While this level of accuracy is sufficient for the original path planning model to deliver a performance comparable to DroNet, in model with LSTM layers, a classifier's prediction depends heavily on the quality of its predecessor prediction. Therefore, if a classifier produces a poor prediction, it can deteriorate the prediction following it.

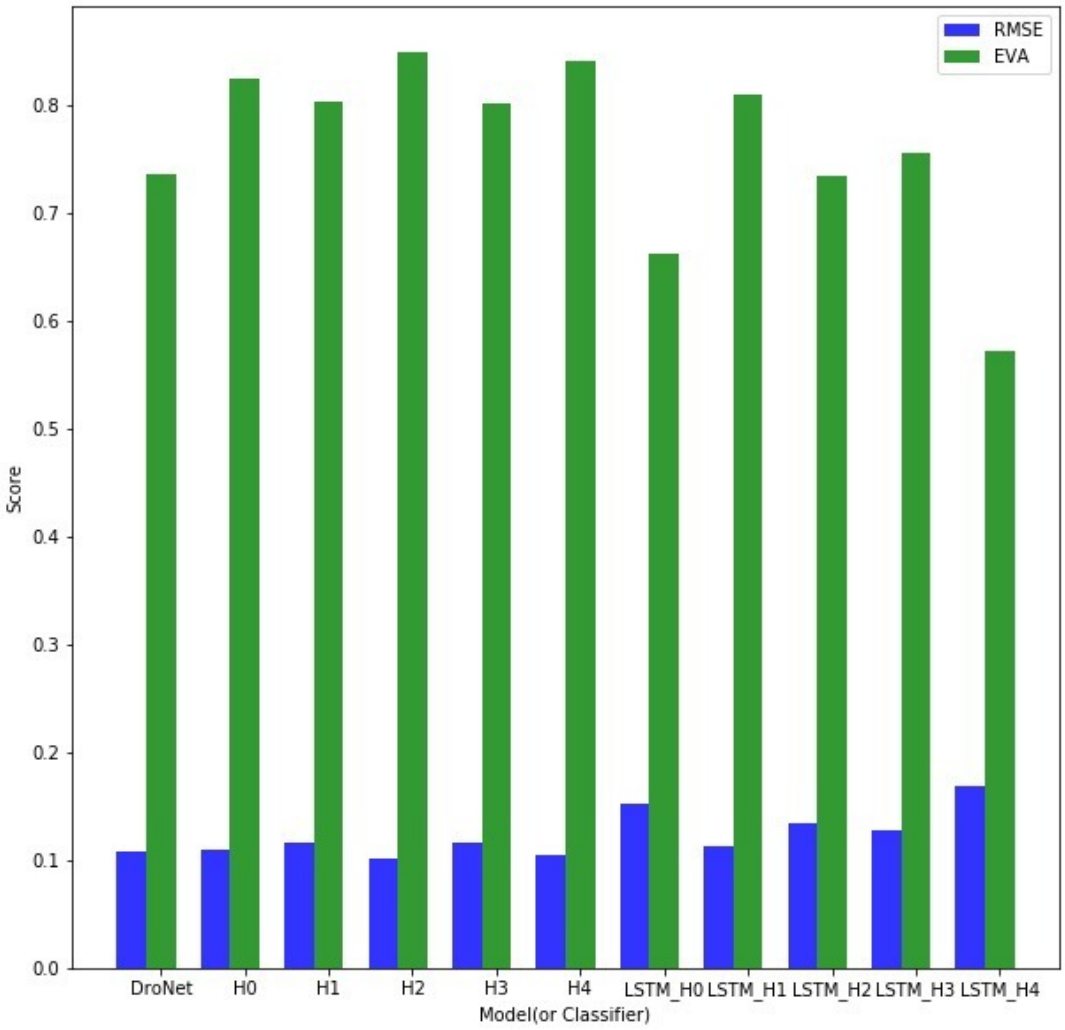


Figure 4.6: Comparison of all model performance on RMSE and EVA metric

Chapter 5

Conclusions

In this project, we have explored an end-to-end learning approach to path planning for autonomous vehicles. In details, a neural network made of three ResNet blocks and an array of classifiers is trained to output a sequence of steering angles which is later interpreted into a geometrical path.

The advantage of this approach is that it pays the way for the integration of deep neural network into the motion planning framework. Specifically, the geometrical path learned by the network is then timestamped using trajectory optimization technique to finally produce a parameterized path (a path with a velocity profile). Beside providing two crucial control signals of the driving task (steering angle and velocity), this integrated approach enhances the reliability of the predicted trajectory while ensures it is natural and consistent with vehicles' kinematic.

For the future work, the accuracy of the array of classifiers needs to be improved. Furthermore, a suitable optimization algorithm needs to be implemented.

5.1 Improve performance with image normalization

Inspired by the fact that Batch Normalization can help accelerate the training process and improve models' performance by getting closer to the minima, the training images are normalized as following before used to train our model.

1. Divide image's intensity for 255
2. Subtract image to the mean of every image in training dataset

The evolution of loss and accuracy of our model during training process are shown in Fig.5.1 and Fig.5.1. As can be seen from these figures, the model needs only 18 epochs to achieve more than 77% of accuracy, which is nearly one-third of previous training time and 5% of accuracy improvement.

As result of accuracy improvement, model's performance (Table 5.1) on on RMSE and EVA are also increased.

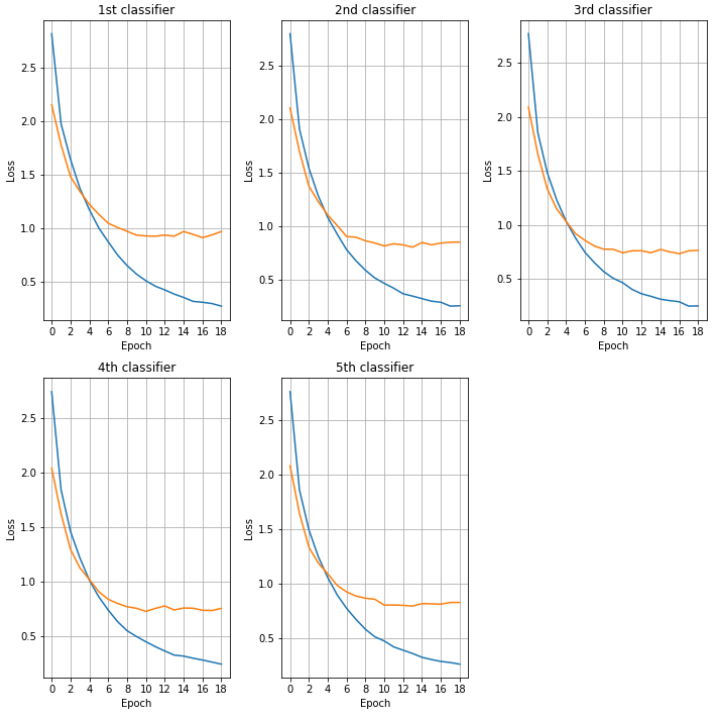


Figure 5.1: Evolution of loss function of each classifier on the training set (blue line) and validation set (orange line).

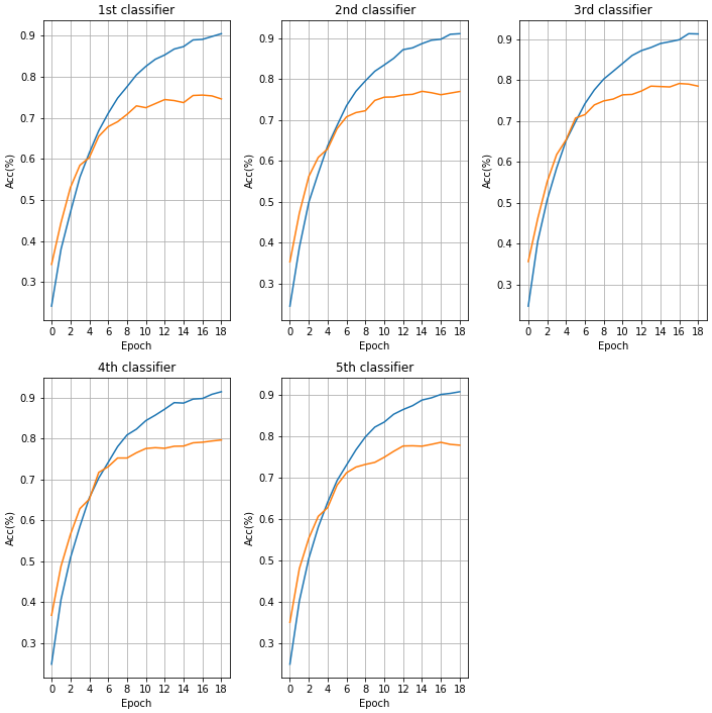


Figure 5.2: Evolution of accuracy function of each classifier on the training set (blue line) and validation set (orange line).

Model	RMSE	EVA	RMSE	EVA
Rnd. b.line	0.3 ± 0.001	-1.0 ± 0.022		
Cnst. b.line	0.2129	0		
DroNet	0.109	0.737		
Head_0	0.1101	0.8245	0.1062	0.8369
Head_1	0.1159	0.8042	0.1059	0.8365
Head_2	0.1012	0.8491	0.1112	0.8177
Head_3	0.1159	0.8020	0.1114	0.8169
Head_4	0.1050	0.8409	0.1189	0.7959
Normaliz.	No	No	Yes	Yes

Table 5.1: Updated (in gray) quantitative performances

OpenSource code The code used in this project is hosted on GitHub in Quan Dao's ECN–E2E repository (github.com/quan-dao/ECN-E2E).

References

[1] P. Varaiya, "Smart cars on smart roads: problems of control," *IEEE Transactions on Automatic Control*, vol. 38, pp. 195–207, Feb 1993. 1

[2] C. Katrakazas, M. Quddus, W. Chen, and L. Deka, "Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions," *Transportation Research Part C: Emerging Technologies*, vol. 60, pp. 416 – 442, 2015. 1, 2, 3, 4, 5, 6

[3] U. Lee, S. Yoon, H. Shim, P. Vasseur, and C. Demonceaux, "Local path planning in a complex environment for self-driving car," in *The 4th Annual IEEE International Conference on Cyber Technology in Automation, Control and Intelligent*, pp. 445–450, June 2014. 4

[4] J. Schroder, T. Gindele, D. Jagszent, and R. Dillmann, "Path planning for cognitive vehicles using risk maps," in *2008 IEEE Intelligent Vehicles Symposium*, pp. 1119–1124, June 2008. 4

[5] J. Ziegler and C. Stiller, "Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1879–1884, Oct 2009. 4

[6] J. M. Wille, F. Saust, and M. Maurer, "Stadtpilot: Driving autonomously on braunschweig's inner ring road," in *2010 IEEE Intelligent Vehicles Symposium*, pp. 506–511, June 2010. 4

[7] M. Bojarski, D. Del 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," *CoRR*, vol. abs/1604.07316, 2016. 6, 7, 8, 16

[8] Y. Lecun, E. Cosatto, J. Ben, U. Muller, and B. Flepp, "Dave: Autonomous off-road vehicle control using end-to-end learning," Tech. Rep. DARPA-IPTO Final Report, Courant Institute/CBLL, http://www.cs.nyu.edu/~yann/research/dave/index.html, 2004. 6

[9] L. Caltagirone, M. Bellone, L. Svensson, and M. Wahde, "Simultaneous perception and path generation using fully convolutional neural networks," *CoRR*, vol. abs/1703.08987, 2017. 7

[10] C. Hubschneider, A. Bauer, J. Doll, M. Weber, S. Klemm, F. Kuhnt, and J. M. Zllner, "Integrating end-to-end learned steering into probabilistic autonomous driving," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–7, Oct 2017. 7, 9, 10, 13, 16, 20

[11] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pp. 39–43, Oct 1995. 9

[12] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, vol. 5, pp. 4104–4108 vol.5, Oct 1997. 9

[13] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, pp. 157–166, March 1994. 10

[14] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Y. W. Teh and M. Titterington, eds.), vol. 9 of *Proceedings of Machine Learning Research*, (Chia Laguna Resort, Sardinia, Italy), pp. 249–256, PMLR, 13–15 May 2010. 10

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. 10, 11, 12

[16] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," tech. rep., Citeseer, 2009. 11

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," *CoRR*, vol. abs/1603.05027, 2016. 11, 12

[18] A. Loquercio, A. Maqueda, C. del-Blanco, and D. Scaramuzza, "Dronet: Learning to fly by driving," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1088–1095, 2018. 13, 16, 19, 24, 34

[19] P. Penkov and V. S. J. Ye, "Applying techniques in supervised deep learning to steering angle prediction in autonomous vehicles," 2016. 13, 17

[20] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," *eprint arXiv:1411.4734*, 2014. 17