Ain Shams University
Faculty of Computer Information Sciences
Scientific Computing Department

Steering angle and Speed prediction for self driving car

documentation submitted as required for the degree of bachelors in Computer and Information Sciences

By Ahmed Araby Kamel Mohamed [Scientific Computing]

Under Supervision of

[Dr/ Maryam Nabil Al-Bery]
[Professor Assistant at Scientific Computing Department]
Scientific Computing Department,
Faculty of Computer and Information Sciences,
Ain Shams University.

[TA/ Basel Safwat]
[Teaching Assistant at Scientific Computing Department]
Scientific Computing Department,
Faculty of Computer and Information Sciences,
Ain Shams University.

August 2020

Acknowledgements

All praise and thanks to ALLAH, who provided me the ability to complete this work. I hope to accept this work from me. I am grateful of my parents and my family who are always providing help and support throughout the whole years of study. I hope I can give that back to them. I also offer my sincerest gratitude to my supervisors, Dr. Maryam Nabil Al-Bery and T.A. Basel Safwat who have supported me throughout my thesis with their patience, knowledge, and experience. Finally, I would thank my friends and all people who gave me support and encouragement.

Abstract

The project aims to develop a deep learning model that drive a car in high speed roads which will help people to use their time in more useful way than driving, also this could help people with disabilities, to solve this problem different approaches were investigated, classical approach (primitive) steering the car based on the geometry of the road lane lines using line detection algorithms i.e. Hough line algorithm, beside resulting in very poor performance there no speed prediction achieved, deep learning approach (state of the art) by developing a model that predict the steering angle and speed using the images of the scene of the road that comes from a static camera mounted in front of the car to achieve second level of autonomy for a self-driving car, Udacity self-driving car simulator is used to test the performance of the model in real time.

Contents

Chapter 1

Introduction

1.1 Problem Definition

autonomous driving is about giving the car the ability to drive itself as humans do, this require a computer to decide the right steering angle of the car, speed, breaks, and to build some sort of self-awareness of the surrounding environment including (people, other vehicles, obstacles, traffic lights, etc.), and most of the tasks in the self driving car stack depend on the scene in front of the car as input to work on. this problem is one of the toughest problems in our century because of the variety of systems and components that should interact with each other to do the task, also, the environment is very complex as there is a lot of variables and unpredictable situations that the system must deal with, doing all this with the safety of the passengers and other pedestrians in mind is very challenging.

1.1.1 History

autonomous driving research started since the trials of Nvidia before the deep learning era using classical computer vision techniques Only to steer the car using lane line detection algorithms, then with the rise of deep learning the research in this area revolutionized, we saw DARPA challenges in (2004, 2005, 2007) to encourage the research more in this area, and now we have different deep learning algorithms and models that try to approach the problem i.e. (end to end learning to predict the steering angle, speed) models that work on LIDAR data and object detection models that provide the self-awareness of the surrounding environment.

1.1.2 Applications

self-driving cars technology have many applications e.g. self-driving taxi which will provide safety and more reliability, trucks, personal car, these examples will help in saving time for people as driving for long distances will not require driver full attention, also people with disabilities will be able to use cars without much help just turn the autonomous driving mood on. self-driving car technology is very promising because of the amazing features and opportunities it makes possible.

1.2 Motivation

self driving car technology will help to make cars more accessible, reliable and safe,

also it will open a door for new industry, which mean more software engineering jobs.

1.3 Objectives

As stated, before self-driving car stack include many systems to interact with Each other to address the whole problem, but this project addresses the part of steering angle prediction and speed prediction, thus the objectives of this project is to :

1. predict the steering angel to keep the car on the road on high ways
2. predict the speed of the car
3. deploy previous 2 features on "Udacity self driving car simulator"
4. deploy the previous 2 features on a small toy car
   however this part is not complete this work will try to share the experience gained during the attempt to work on real small toy car and discuss the problems involved.

1.4 Time Plan

- Planning:
  started at 10-oct-2019 and ended at 25-oct-2019,
  this period was for investigating the problem history, current state of the art approaches and the skills needed to complete the task in hand e.g (steering angle prediction, speed prediction and hardware components needed and interfacing with theses components)
- Survey:
  started at 30-oct-2020 and ended at 18-Nov-2019,
  this period was for reading papers and collecting more information about the state of the art techniques used to approach the problems that this work is interested in, also a summary for the some papers was created.
- Analysis: starts at 19-NOV-2019 and ended at 18-Dec-2019.
- design: starts at 20-Dec-2019 and ended at 23-Jan-2020.
- Implementation: starts at 24-Jan-2020 and ended at 3-March-2020.
- Testing: starts at 4-March-2020 and ended at 2-May-2020.

1.5 Documentation Outline

this thesis is composed of 7 main chapters, Chapter one (this is where you are now and you know what it's about), the Second Chapter is Background, in this chapter I will talk about a literature overview and Summary explanation for the topics that need to be covered to keep going in this work also I will discuss the survey done before this work started.

the third chapter is System Architecture in this chapter I will discusses the architecture design of the architectures used in this work and and explanation for the technical decisions and choices that are made.

the forth chapter is System Implementation in this chapter I will discusses input and output preparation, preprocessing and augmentation also I will discuss the choices of the training parameters of the model finally I will show the results.

the fifth chapter is System Testing in this chapter I show show how to test the model in the udacity simulator for self driving cars and how the model and the simulator interact with each other.

the sixth chapter is a discussion for my attempt to build a small toy self driving car to deploy the model on it, also I discuss the mistakes I made.

seventh chapter is Conclusion and Future work in this chapter I will discusses advantages and disadvantages of my the project from my perspective, and the future works on this project that I will do.

at the end there will be sections for list of abbreviations the tools used and references for the resources and papers, that helped me out in this project.

Chapter 2

Background

in this chapter I will discuss the survey about the autonomous driving problem focused at steering angle and speed prediction, then I will move to the literature review to give a quick summary for the topics and skills needed to start working on this problem.

2.1 approaches to solve the problem

to solve the task of steering angle prediction and speed prediction there is a classical approach using image processing and computer vision, Artificial Intelligence (deep learning approach using end to end learning or Reinforcement Learning), this work follow the deep learning end to end approach, hence I will introduce the concepts needed for this approach

2.2 Artificial Intelligence

Artificial Intelligence is the science that enable computers to accomplish some task like humans, and theses are kind of task that are nearly impossible to be solved using rule based programs.

to build a normal program what ever this program is supposed to do it will be rule based, series of rules that describe an algorithm to get the output, theses rules are designed based on:

- available input
- analysis of the problem

on the opposite side to develop an Artificial Intelligence model we give the model some examples of the available input and the corresponding output,

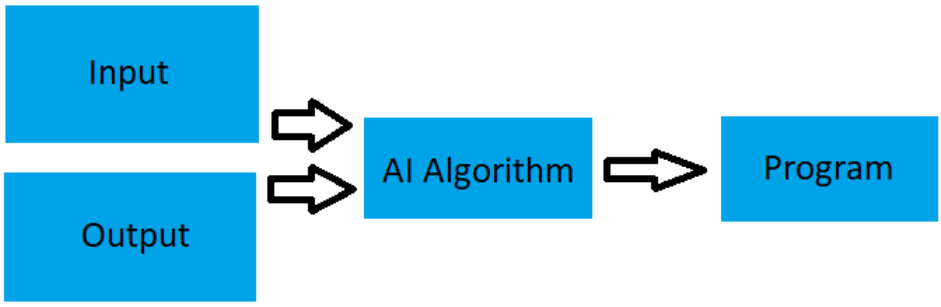


Figure 2.1: building Artificial Intelligence Model

then using an AI Algorithm we get the model that represent the program, then this model could take input and produce output and this processes is illustrated in figure 2.1

2.3 Neural networks and Deep Learning

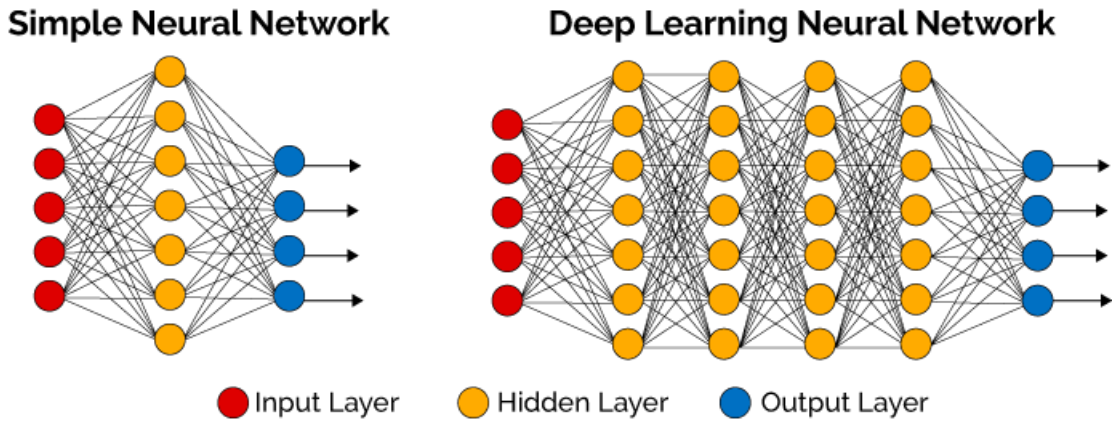


Figure 2.2: Neural network structure

neural networks are subset of machine learning that is by it's turn a subset of Artificial intelligence, Neural networks try to estimate the function or the rule that will help in solving the problem in hand, the structure of Neural Networks is :

- Input Layer
- Hidden Layer
- Deep Layer

there is 2 types of neural networks

- Shallow NN small number of hidden layers
- Deep NN big number of hidden layers

the more hidden layers the NN have the more complex functions it can represent hence, solving the problem efficiently.

2.4 CNN

Convolutional neural network is a special type of Neural networks, it works on images or high dimensional data that have spatial relation between it's components, it is based on convolution, weights that the network learns are located on the kernels(filters),

it has the benefit of local connectivity and sharing weights, they work for reducing the number of weights involved in the learning operation under the assumption that important features could be extracted from small local areas instead of all the image at once, which are very effective in dealing with high dimensional inputs like images, this will reduce the possibilities of over fitting.

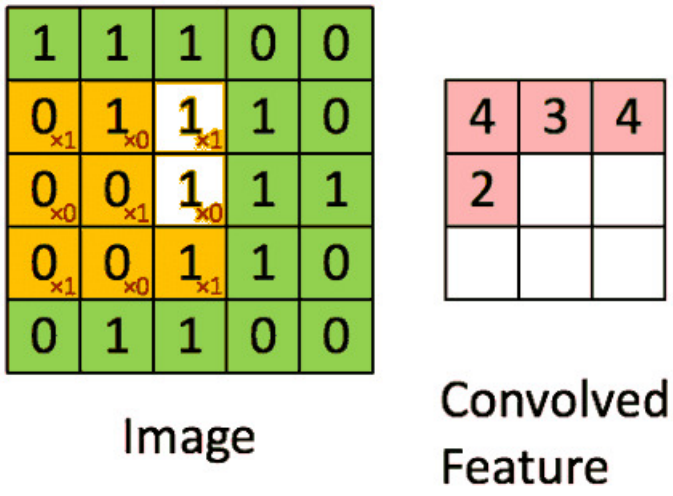
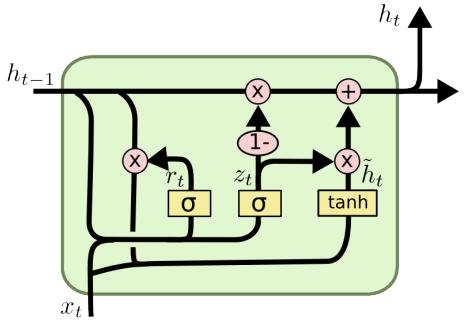


Figure 2.3: CNN layer, image has green color, filter has the orange color and it's overlaid on the image

2.5 LSTM

Long Short Term Memory is a special type of recurrent Neural Networks it's used to capture the long and short term temporal dependence between input instances.

capturing the temporal dependence is very important to the speed prediction task, to predict the current speed we can't relay only on the current input of this moment, but we also need the history of the car speed over a specific period of time, LSTM is able to combine theses information and produce the best possible prediction for the speed.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 2.4: LSTM layer with the inner gates illustrated

LSTM works throw some components called gates, they are a small neural networks inside the lstm layer. they are all combined together works on deciding how mush information to keep from the previous history of the input instances and how mush information is used from the current input.

specific role of each gate could be understood from the input and the activation layer of the gate.

2.6 end to end learning

end to end Learning is a technique used to train Deep Neural networks it is most used with complex problem e.g (problems involving computer vision stuff), as we stated before most of the time the specific features that the model need to learn the task in hand is not well know, and this technique enable the model to choose the features that help in the learning processes.

in our case e.g(dealing with input images) just feed the model with the input image and it will learn to extract the best features to apply the regression or classification task on them, without any human intervention or hand

crafted features.

it works best with deep networks as different group of layers will be specialized in extracting different types of features.

2.7 PID Controller

PID stands for proportional–integral–derivative, as cars can only accelerate or decelerate not to move from a speed to another one instantly. we need the PID Controller to put the car at the desired speed, having the predicted speed and the current speed of the car using the PID Controller we can calculate the error and calculate the value of acceleration or deceleration

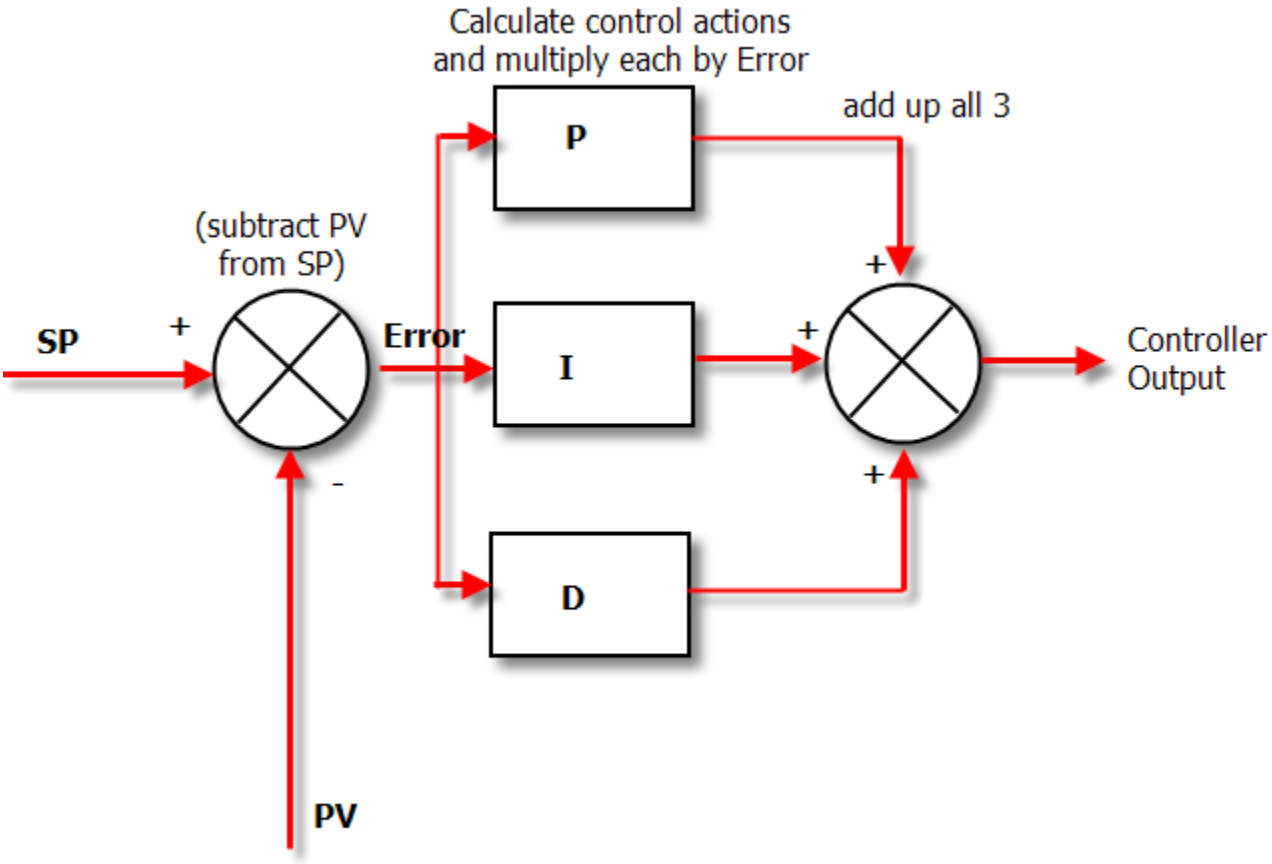


Figure 2.5: PID Controller

2.8 survey

after the brief knowledge discussed in the literature preview section, survey discussion could be better understood.

2.8.1 classical approach

before deep learning era, attempts to get a car to steer it self was carried on using image processing and classical computer vision techniques it was all about lane line detection (i.e hough line transform algorithm) with additional techniques to build some understanding of the part of the road the car is currently at using the geometry of the detected lane lines, hence the car (e.g the computer in the car) get to decide the right angle that it should steer on it's very obvious how important the lane line to this approach which could be considered as a weakness point, for sure lane line is not the only thing that help people drive cars, there is another factors and variables that should be taken into account to have a car that drive it self like humans do, but the problem is we don't know theses factors and variables, also lane lines could not be perfectly painted on the road which would cause a problem for this technique as all what it depends on is the lane lines of the road.

2.8.2 Deep Learning approach

as we stated before in the classical approach part, the main issue was the factors/feature that the techniques depends on to decide the angle, also there was no learning the whole thing was rule based using some mathematical model that work on the geometry of the road lane lines, but this paper paper reference used deep learning with end to end technique to train a model that will predict the steering angle of the car based on the feature it extracts from the scene coming from the front of the car using a static camera mounted on the car, and this end to end technique was fascinating as the model learns the features it needs to accomplish it's task, which mean it could figure what is important beside the lane lines of the road, hence results starts to get better

Chapter 3

System Architecture

in this chapter I will show the main Architecture used to predict the steering angle and speed of the car also I will break it into smaller components that do different jobs and explain the inner working of it, then I Will talk about Nvidia architecture and compare it's results with the main architecture used in this work.

at the end I will illustrate how does the model talk to the Simulator to drive the car during testing.

3.1 End-to-end Multi-Modal Multi-Task architecture main model used in this work

this model is the main model used in this work it's called multi model multi task as it solve the two problems e.g. (steering angle prediction and speed prediction) at once.

multi model multi task architecture idea is to combine two models to work as one and to use intermediate features of the two models to help each other in the prediction for the different tasks under the assumption that tasks are related in some sense and both require the extraction of the same features form the input.

multi loss cause the error in one task to affect the other task as error is combined in the Backpropagation which play the role of regularization that is used to reduce the loss but in more effective manner as this force the earlier layers to extract the most important features that satisfy both tasks. as mentioned here [14], [15].

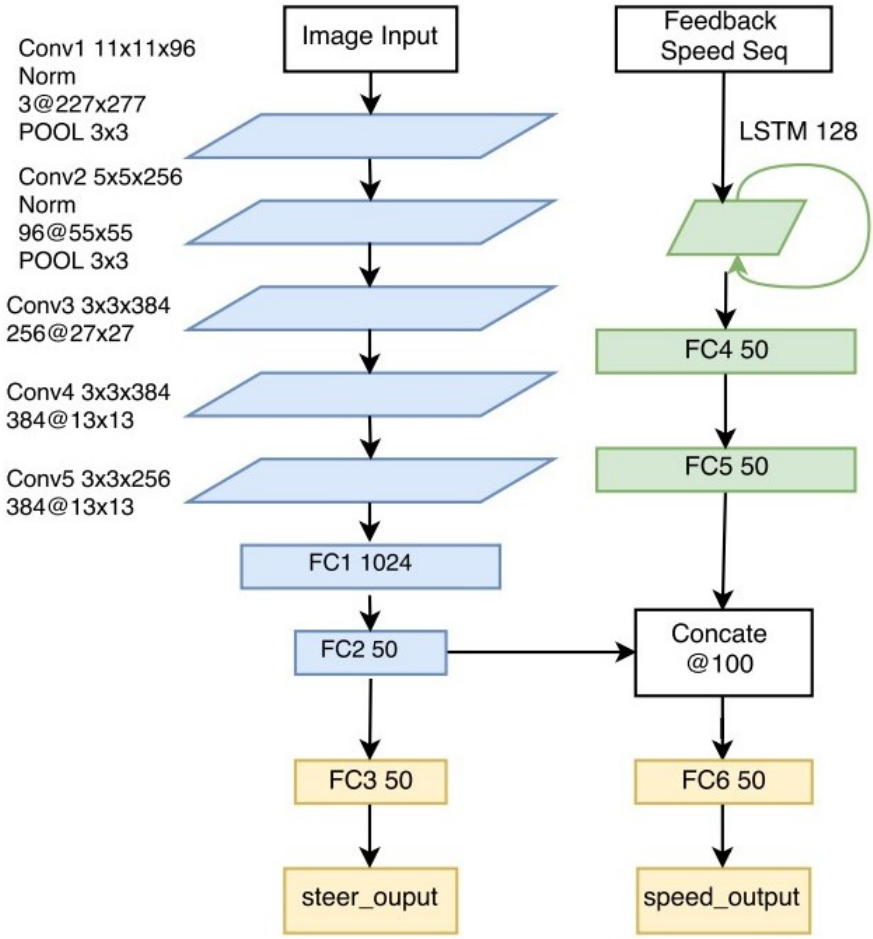


Figure 3.1: End-to-end Multi-Modal Multi-Task Architecture

3.1.1 angle prediction

as said that the main architecture is multi model, here I will illustrate the part concerned about the steering angle prediction it takes as input an image of the scene from the camera mounted in front of the car.

it has 5 Convolutional layers first 2 are followed by batch normalization layers and max pooling layers.

last 3 Convolutional layers are just CNN layers with no additional layers.

then we have Fully connected layer that is responsible for the regression part that has 3 fully connected layers.

output layer produce normalized angle in the range [-1, 1] -1 is most left 1 is most right then the normalized angle could be mapped for the angle range of the simulator or the car in hand.

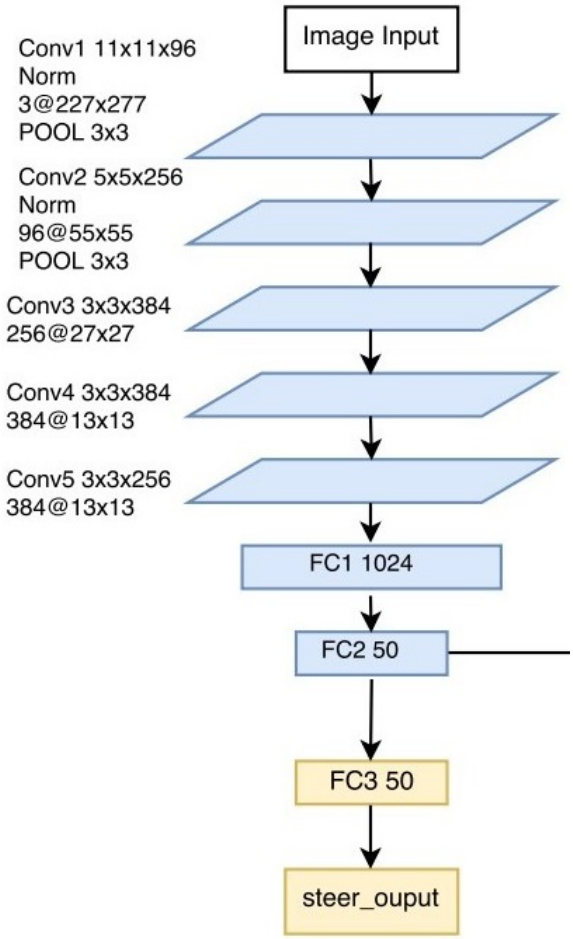


Figure 3.2: Angle model of the End-to-end Multi-Modal Multi-Task Architecture

3.1.2 speed prediction

speed prediction part of the multi model network take as input a sequence of the previous speed values in last 15 frames and here is the role of LSTM layer to combine theses values and capture the temporal dependencies between theses values, the LSTM part have 128 neuron in it's hidden layer

then we have 2 fully connected layers that work on the output of the LSTM layer to capture more features.

more explanation for the last part of the speed prediction model e.g.("Concate") will be discussed in the next section.

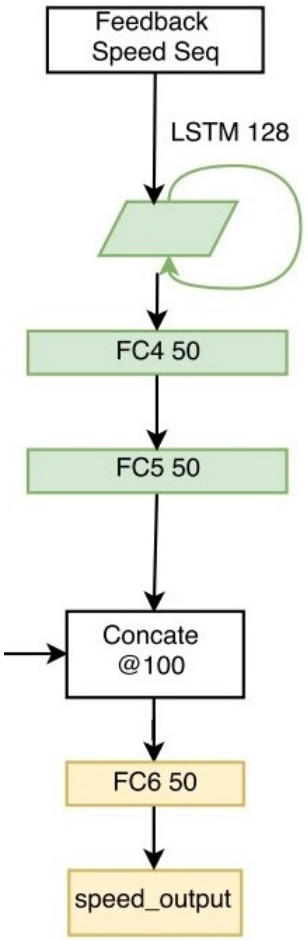


Figure 3.3: Speed model of the End-to-end Multi-Modal Multi-Task Architecture

3.1.3 Combining visual features and speed sequence

concatenate layer that combine the intermediate visual features from the steering angle model and the speed sequence features to build feature vector that the speed model will use to predict the speed value.

the combination of visual features and speed sequence features is considered as strength point in the "multi model multi task architecture" as this mean that error in speed value prediction will Backpropagate from the speed model to the steering angle model, forcing the steering angle model to extract features that are expressive for both tasks i.e (angle and speed prediction)

in the original paper [3] beside the final version of multi model multi task architecture that I use in my work here, they also tried different architecture to achieve the multi loss functionality, one of them was to predict the speed based on the same exact input and features of the steering angle prediction which is just the images of the scene in front of the car without taking into account the speed sequence history of the car, other work was solving the two problems as two completely separate models (tasks).

the former found that taking the speed sequence into account enhance the prediction and reported that using multi model multi loss help in beating the over fitting, the latter faced over fitting problem.

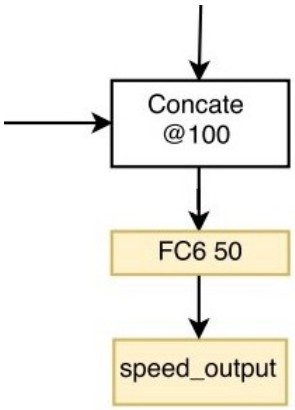


Figure 3.4: concatenating the visual features and the speed sequence features

3.1.4 why LSTM for speed prediction ?

using the previous speed values as extra modality in addition to the visual features for the speed prediction could give the model more understanding about the part of the road that car was at i.e. if the car was slow for most of the previous speed value it's probably because there was a turn so the model might keep the speed low for now even if the visual features don't capture the turn any more which could happen if the car is very close to the turn as small parts of curvature could appear as lines.

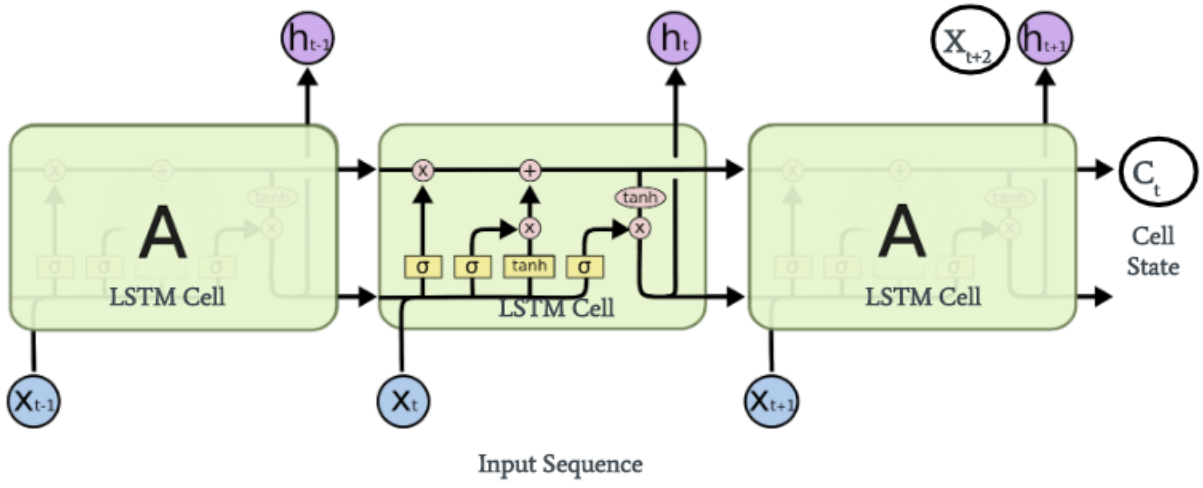


Figure 3.5: concatenating the visual features and the speed sequence features

3.2 Nvidia Architecture

Nvidia architecture [1] only predict the steering angle and it is quite simple, as part of this work I trained this architecture and tested it on the Udacity Simulator to compare it's results on the steering angle prediction to the results of the main architecture my work i.e (end to end multi model multi loss for steering angle and speed prediction) omitting the speed prediction from the comparison.

the reason that I present this architecture in my work is to investigate the claim that multi task models have the advantage of generalization over models with single task for the same problem,

also I used the technique they used for collecting their training data to build a trainer application that collect training data for the small toy car that I tried to build but couldn't due to some issue in the chassis part of the car and decide to terminate this part of my work then, latter in coming chapter I will present the work I did during my attempt to deploy the model in a small toy car.

the architecture is relatively simple as shown in figure 3.6:

- 5 layers of convolutional layers for feature extraction first 3 have 5X5 kernals last 2 have 3X3 kernals
- then we have fully connected layer for the regression task to predict the steering angle 3 fully connected layer followed by the output layer

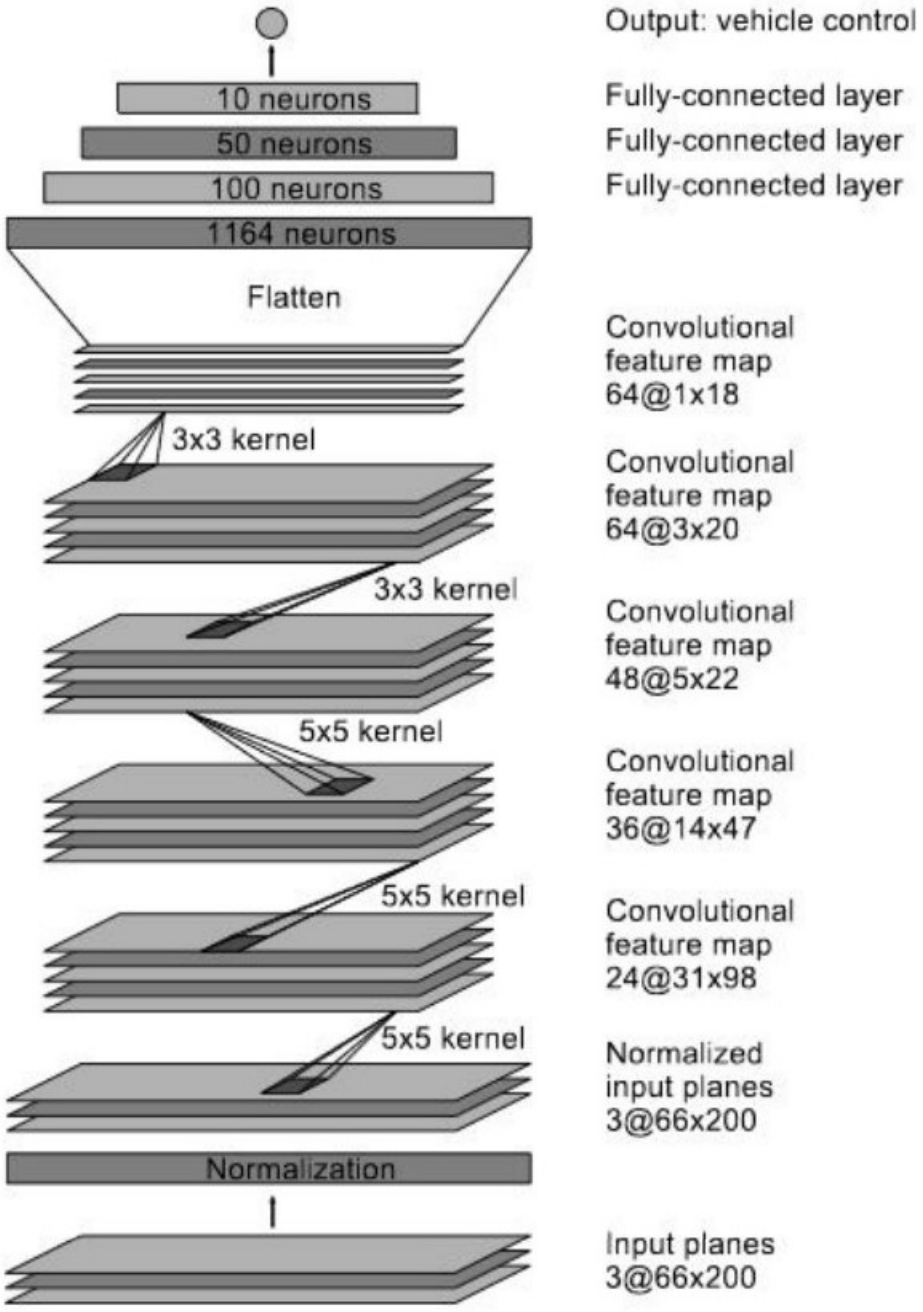


Figure 3.6: concatenating the visual features and the speed sequence features

3.3 Simulator

for testing the model prediction for steering angel and speed I used an open source simulator created by Udacity for self driving car [10] it enables me to test the steering angle and the speed values coming form my trained model, the simulator track was similar to high way roads but there was no additional cars around, and this was good point for me as my goal from this work is to build the steering angle and speed prediction part, and if there was another car this would need the object detection part to avoid crash with other cars.

but at the same time the track was very challenging it has many high hills, low cliffs and very sharp turns, which was a very good measure of generalization in the model learning.

Figure 3.7: Screen shoot of the udacity simulator track

Chapter 4

System Implementation

4.1 System overview

4.2 Data Set

data set used to train the model is collected using the Udacity self driving car simulator [10] at the beginning I used a public data set published by Udacity [**?**] but as the model I use have large number of learned parameters and the data set was small and it didn't have sufficient variation for the situations that could face the car my model was overfitting the training data and have bad performance on the validation data and even worse on the simulator during the final testing.

so I had to collect the data set myself [13] as the simulator have training mode to collect the data set and I tried to go into many scenarios as possible, failure cases and recover cases.

the data set collected is:

- images of the scene in front of of the car from left , center and right cameras
- steering angle from the center camera perspective, normalized in range [-1, 1] = [most left, most right]
- acceleration, breaks.

Figure 4.1: Screen shoot illustrate the different modes of the simulator

4.3 preprocessing

in this section I will discuss the input and output preparation for the multi model architecture

the convolution layers input take 1 RGB image of the current scene in front of the car from the center camera , and the LSTM layer take speed sequence which represent the speed values for the previous 10 frames before the current one, this represent 1 instance of the input for the Network.

for the output part, each frame is attached with it's corresponding angle value and each speed value sequence is attached with the right speed value that should be predicted for the current frame.

applied preprocessing is :

- image resizing as the simulator produce 320 X 160 , width and height in pixel respectively and the Architecture take input of size, 320 X 70, most important features to keep are in the horizontal dimension, that is why the input images keep their original width, but the height is scaled down to reduce the computation and model complexity to avoid overfitting.
- also the image pixel intensity is normalized to be in the range [0, 1] this helps the model to learn faster.
- normalizing the speed values to be in the range [0,1] originally the range was [0, 30] mph, but this pushed the model to be biased toward reducing the loss of the speed at the expense of the angle prediction as angles values are already normalized which mean they produce small loss, this step was very important to get stable results for both angle and speed prediction from the multi model multi task architecture

Figure 4.2: before preprocessing.

Figure 4.3: after preprocessing.

4.4 Data Augmentation

in addition to collecting the data set myself and trying to generate possible variations of the situations that could face the car, this was not enough to beat the overfitting problem, but augmentation techniques helped very much to achieve some sense of generalization.

in this section I will discuss the techniques used for data augmentation with visual examples.

4.4.1 Left and Right Camera Calibration

Udacity self driving car simulator record the scene in front of the car using 3 cameras positioned in different places on the car, left, center, right of the driver view, but only recording the angle corresponding to the center camera, corresponding to the view that the driver make decision on it, hence we can't train the model using the frames from left and right camera using the same angle.

Figure 4.4: View from center camera, driver view angle = 0.5728891

Figure 4.5: View from right camera angle = 0.5728891 - 0.25

Figure 4.6: View from left camera angle = 0.5728891 + 0.25

$$\theta_f = \theta_r + \arctan(\frac{d_y}{s * t_r})$$

Figure 4.7: Synthesis equation.

so to get the right angles for left and right frames we use the equation in figure 4.7. from [3]

4.4.2 Horizontal flipping

we could increase the variation in our data set using horizontal flipping, but then we need to flip the sign of the angle this is better illustrated in figure 4.8.

as the green vector represent the direction of car movement, then by flipping the frame this will also flip the vector that represent the car movement direction , and it will be the blue vector

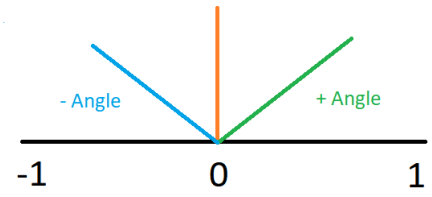


Figure 4.8: Angle Flipping visualization.

Figure 4.9: original frame before horizontal flipping.

Figure 4.10: frame after horizontal flipping.

4.4.3 Adding Shadows

also adding shadows will simulate challenging situations, and this technique will help the model to be more robust.

the code used to generate the shadows guarantee that:

- Intensity of the shadows is randomly selected
- Position of the shadows is randomly selected

Figure 4.11: Frame without Shadows.

Figure 4.12: Frame with Shadows.

4.5 training parameters

parameter	value	comment
inputH	100 pixel	
inputW	320 pixel	
inputC	3 (RGB)	
speed sequence length	10	number of speed values from the previous frames
angle shift center	0	
angle shift right	0.25	
angle shift left	-0.25	
FC drop out	0.5	
Learning rate	0.0001	then it get reduced to 0.00001
Spliting precentage	0.01	
Batch size	256	
Num epochs	30	

4.6 architecture details

for second and third CNN layers padding property = same which mean the feature map result will have the same dimensions of the input for each layer.

activation functions and loss function for all the layers in multi model multi task architecture the activation function is "elu" and the loss function for both tasks is "mean square error" and both task have the same weighting Combined.

to reduce the computation of the training processes and model complexity max pooling layers were added

to achieve generalization drop out and regularization techniques were applied.

more details on the architecture design in figure 4.13

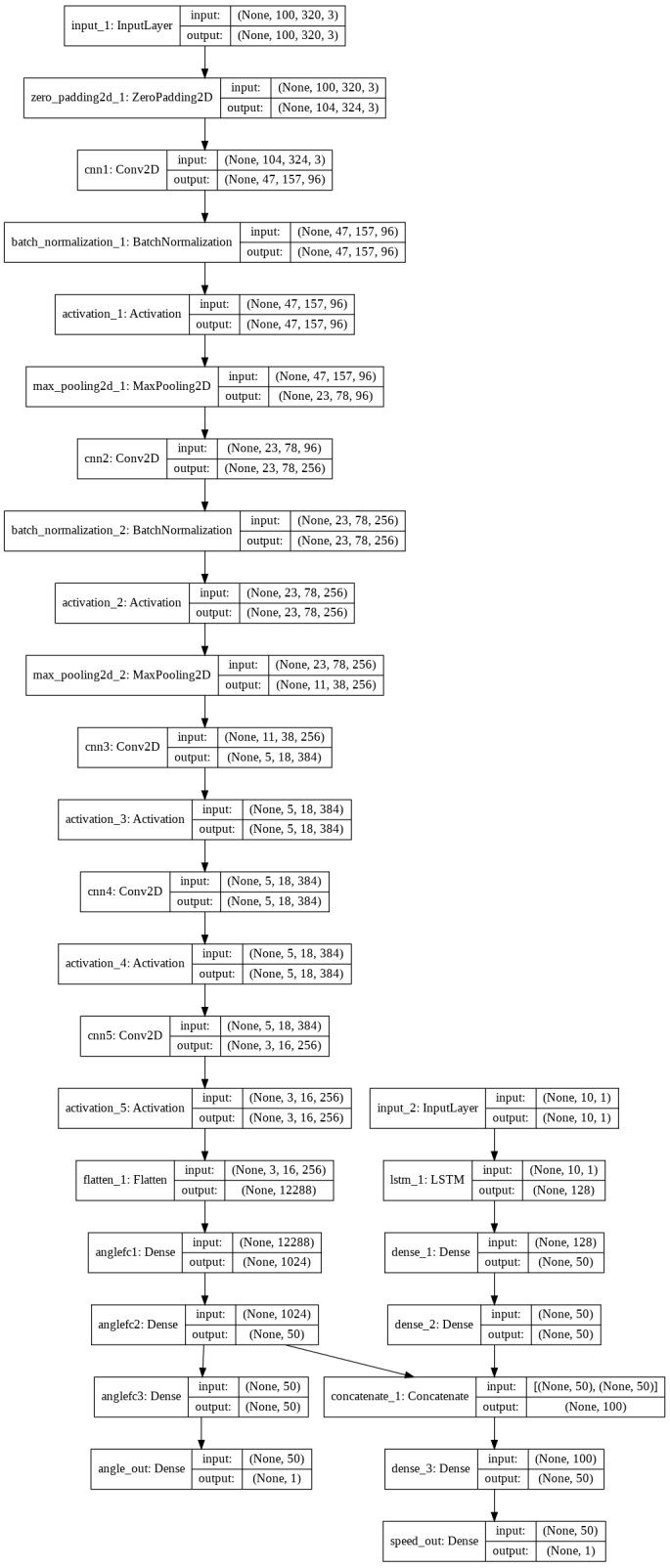


Figure 4.13: Details for multi model multi task architecture.

4.7 results

Loss values for the main architecture "end to end multi model multi loss Architecture"

–	angle loss	speed loos
training	0.0790	0.002
validation	0.07612	0.002

Table 4.1: Training and validation loss

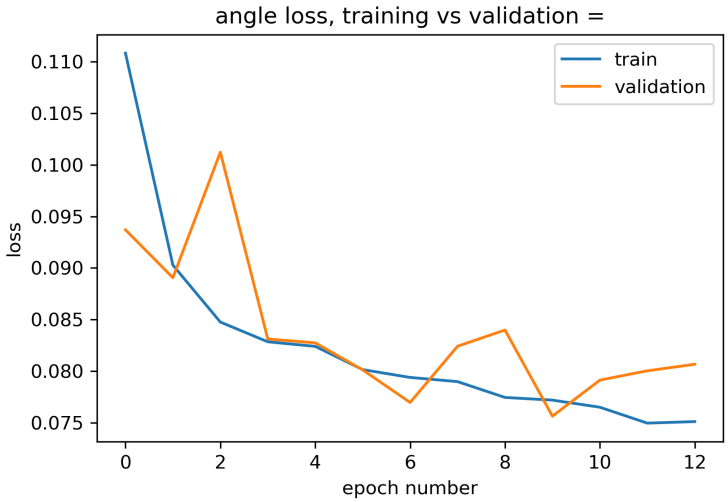


Figure 4.14: angle loss, train vs validation.

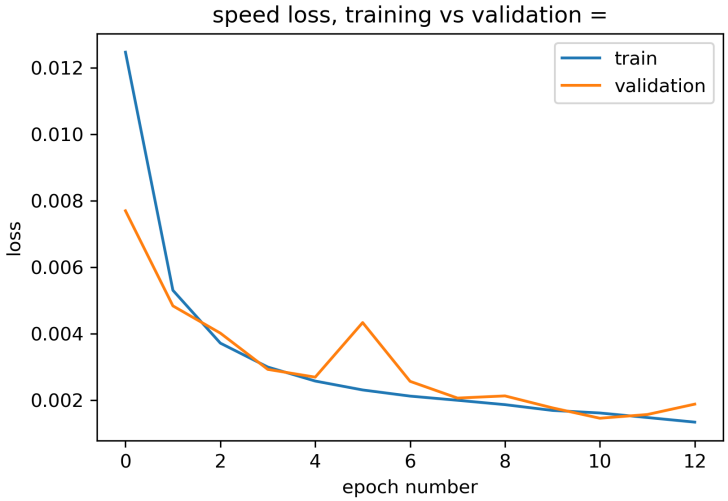


Figure 4.15: speed loss, train vs validation.

4.8 comparison

here I will compare Nvidia architecture with the main architecture for my work i.e (end to end multi model multi loss for steering angle and speed prediction architecture) only for steering angle loss as Nvidia don't predict the speed value.

The reason for this comparison is to validate the claim that multi model multi loss architectures will have the benefit of regularization and generalization better than architectures that work on one task and use regularization.

as stated here [14], [15] that multi tasks work as regularization for each other, as they share the loss function which force the model to extract the most important features that could satisfy both task under the assumption that the 2 tasks are similar and could be achieved by working on the same extracted features.

Architecture	train loss	validation loos
Multi model multi loss model	0.0320	0.0685
Nvidia model	0.0790	0.7612

Table 4.2: Nvidia angle loss .Vs. Multi model multi task angle loss

results in this table contradict with the claim that multi task learning provide generalization, how ever at testing on Udacity simulator, multi model multi task architecture perform very close to the Nvidia architecture from the steering angle perspective.

Chapter 5

System Testing

for system testing I will use Udacity self Driving Car Simulator. but first I need to show how the model will interact with the simulator in terms of data exchange, then I provide screen shots for the testing processes.

5.1 simulator and model interaction

my model need the frames of the scene in front of the car and speed sequence to start working, and the simulator need to receive the steering angle prediction and speed value in order to move the car, hence we have client-server structure in which my model is deployed in a flask server i.e(web development framework),

and the simulator work as a client that will connect to the server and send continuous stream of the frames from the scene in font of the car and flask server will give theses frames to the model for prediction , then it will respond with the steering angle value and speed value and the simulator will move the car using theses values.

for testing, only the flask server file that is called multi drive need to be executed from the command line with an activated Conda environment that has all the needed dependencies, then start the simulator in autonomous mode to convert the simulator into a client.

Figure 5.1: screen shoot that show how to establish connection with the simulator , the code running in the command line is the flask server that have the mode.

Chapter 6

Hardware this part is not delivered with the project

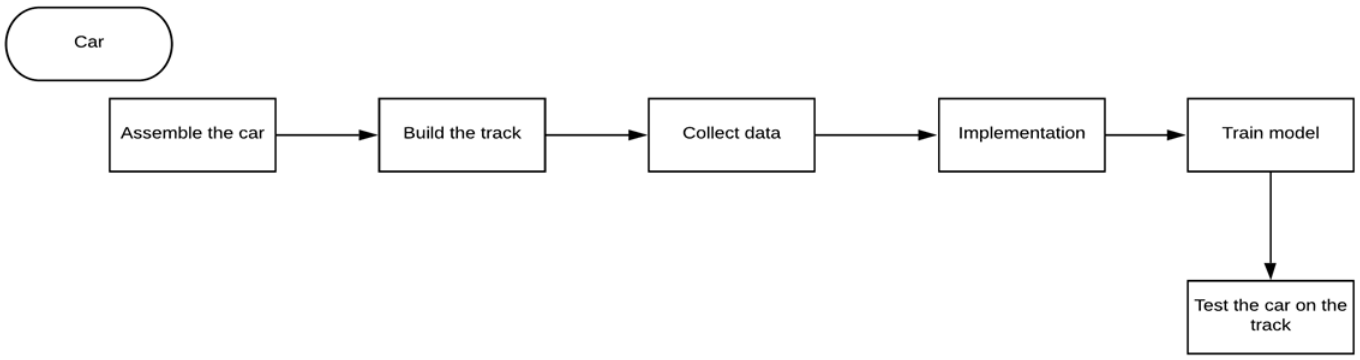


Figure 6.1: steps to build small toy self driving car

this is an abstraction for the steps needed to build the toy self driving car it's similar to the pipe line for deploying the model on a simulator. here in this chapter I will try to discuss the experience I had during the attempt to build the toy self driving car, even if I didn't succeeded in this but still I have mistakes to share with you, so you can avoid doing the same.

also I will give a little push for starting in this track as I will discusses the hardware components needed and I will explain the diagram above in more details.

6.1 components

components needed for the car are electronics components and car chassis, also a laptop is needed to do the computation as will be shown in the next sections.

6.1.1 Hardware

- raspberry pi 3 model B+ or higher it's the brain of the car it controls the motors and every thing in the car also it could do the prediction of the steering angle and speed but it's computation power is weak for doing this at real time so we will delegate this task to a more power full computer.
- raspberry pi camera to capture the scene in front of the car.
- 2 DC motors for the back wheels of the car.
- 1 servo motor for the front wheels of the car as the raspberry pi would convert the predicted steering angel to a PWM so that the servo could physically steer the car.
- L298N Motor Driver Module or L293D Motor Driver so that the PI can control the DC motors.
- batteries, wires big number of hidden layers

Figure 6.2: raspberry pi 3 model B+

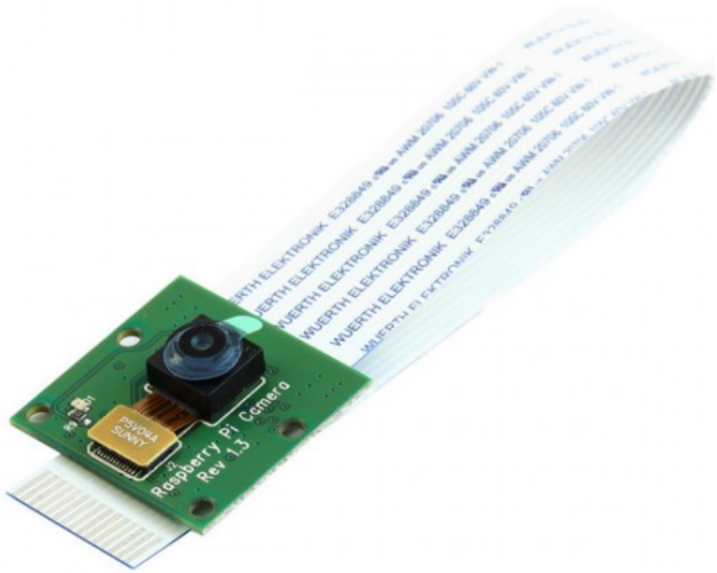


Figure 6.3: raspberry pi camera

6.1.2 car chassis

I can tell that this is the reason that my attempt to build the toy self driving car didn't succeed there is 2 choices here

- buy a car kit that have the car chassis and all the electronics components above including the PI.
- print a 3D one you self dont do this unless you are experienced in design and 3D printing stuff

obviously I took the 2nd choice despite the lack of knowledge in design and 3D Printing because I did have all the electronics components before so I didn't want the full kit, but the 3D printed car chassis didn't fit with all other components very will and this ended up with a lot of editing and printing then terminating the processes, because this was costly, and it exceeded the budget for the project.

6.2 collecting data

as the domain of the small toy car is very different from the domain of the real data sets for steering angle prediction and the domain of the simulator I won't be able to use any trained model on theses domains to drive the toy car so I had to collect the data my self using the trainer Application I wrote here in the project repo, it basically capture the frame and the steering angle performed by the human that play with the car and attach both to each other as training instance then write them in batches to the hard disk.

6.3 system components working together

the PI should control every thing in the car and also it will communicate with a more power full computer i.e (personal laptop is enough), using sockets so that the PI will send to the computer the frame of the scene in front of the car and the computer will feed this frame to the model and produce prediction for the steering angel then send it back to the PI, then the PI will convert the steering angle into PWM and communicate with the servo to physically steer the car. this mechanism is very similar to the one followed in Udacity simulator, but here I had to write the client - server module my self using socket the code is uploaded on the repository of the project it is called Alien Pilot [12]

Chapter 7

Conclusion and Future Work

7.1 Conclusion

7.1.1 summary

self driving car is a very promising technology with a complex stack of systems needed to work together to build a self driving car, in this work I showed one component of this stack which is steering angle and speed prediction, we discovered that both are very related problems and could be solved together using multi model multi task architecture and achieve good generalization.

7.1.2 advantages and disadvantages

- Advantages are:
  - solving the two problems in multi model multi task architecture which helped the model to achieve generalization.
  - contributing with a relatively large data set that have various challenging situations [13]
- Disadvantage are:
  - using a simple simulator for testing, because using another simulator with a complex environment would need additional systems to work with the speed and angle prediction model e.x(object detection for collision detection).
  - speed prediction produce small values in straight roads it would be better if the model manged to go fast in empty straight road.

7.2 Future Work

as I will keep studying the self driving car problem, I want to:

- use more complex simulator with complex environment like carla with more systems deployed other than the angle and speed prediction, i.e (object detection, path planning etc.).
- continue my work on the real small toy car to deploy my work on it.

Tools

7.3 Softwere tools

- python is the main programming language for this project
- keras is an open source deep learning library for building , training and testing the model
- open cv for different preprocesing and augmentation techniques
- matplotlib for result visualization and data set analysis
- numpy
- pandas for data set loading and manipulations
- flask it's the server that receive the frames from the simulator and send the commands for steering angle ans speed back to the simulator.

7.4 Hardware

this is related to the part that is not delivered with the project, but some work was done on it

- raspberry pi
- raspberry pi camera
- DC motors
- servo motor
- 3D printed car chassis
- L298 motor driver

Chapter 8

Appendix

8.1 List of abbreviations

- AI :Artificial Intelligence
- NN :Neural Network
- CNN :Convolutional Neural Network
- RNN :Recurent Neural Network
- LSTM :Long Short Term Memory
- PID :proportional–integral–derivative
- SIM :Simulator
- inputH :Input height in pixels
- inputW :input width in pixels
- inputC :input color space dimension e.g(RGB)

- DC :direct current
- PI :raspberry pi
- PWM :Pulse-width modulation
- repo :github repository

List of Figures

List of Tables

Bibliography

[1] Mariusz Bojarski, Davide Del Testa, et al. "End to End Learning for Self-Driving Cars". Nvidia, 25 Apr 2016

[2] Hesham M. Eraqi, Mohamed N. Moustafay, and Jens Honerz. "End-to-End Deep Learning for Steering Autonomous Vehicles Considering Temporal Dependencies". 22 Nov 2017

[3] Zhengyuan Yang1, Yixuan Zhang1 , et al. "End-to-end Multi-Modal Multi-Task Vehicle Control for Self-Driving Cars with Visual Perceptions". Department of Computer Science, University of Rochester, Rochester NY 14627, USA. 2 Feb 2018.

[4] Rounak Hazra, Aman Kumar, B.Baranidharan. "Effect of Various Activation Function on Steering Angle Prediction in CNN based Autonomous Vehicle System". International Journal of Engineering and Advanced Technology (IJEAT). December, 2019.

[5] Shuyang Du, Haoli Guo, Andrew Simpson. "Self-Driving Car Steering Angle Prediction Based on Image Recognition". Stanford, 11 Dec 2019.

[6] Jingwei Cao, Chuanxue Song. "Lane Detection Algorithm for Intelligent Vehicles in Complex Road Conditions and Dynamic Environments". sensors, MDPI. 18 July 2019

[7] Michael Diodato, Yu Li, et al. "Winning the ICCV 2019 Learning to Drive Challenge" 23 Oct 2019.

[8] Aliasgar Haji, Priyam Shah, Srinivas Bijoor. "Self Driving RC Car using Behavioral Cloning". 10 Oct 2019.

[9] `https://www.csimn.com/CSI_pages/PIDforDummies.html`
Last Retrieved on 12/8/2020

[10] https://github.com/udacity/self-driving-car-sim
Last Retrieved on 12/8/2020

[11] https://rb.gy/mw6afd
Last Retrieved on 12/8/2020

[12] https://github.com/Ahmed-Araby/Alien_Pilot
Last Retrieved on 12/8/2020

[13] https://rb.gy/m1em0r
Last Retrieved on 12/8/2020

[14] https://rb.gy/nkapvv
Last Retrieved on 12/8/2020

[15] https://ruder.io/multi-task/ Last Retrieved on 12/8/2020