Seminar Computer Vision by Deep Learning
CS4245
2019 - 2020

End-to-End Autonomous driving for embedded agents using CNN's

PROJECT GROUP 3
Vibhav Inna Kedege (4998596)
Lokin Prasad (5092396)
July 1, 2020

Contents

1 Introduction

Over the years, Deep Learning has been utilised in various applications. Autonomous driving is one such example. Alongside the development of deep learning techniques in the automotive field for commercial vehicles, there is also research being done on the deployment of such models onto various embedded automotive platforms. Inspired by this, our main motive in this project is to design and compare end to end light deep neural networks that are capable of performing autonomous driving on a predefined track when deployed on a low resource hardware platform.

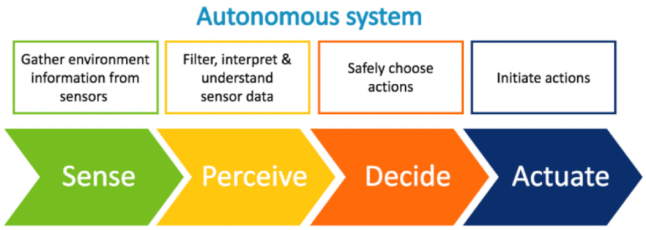


Figure 1: Autonomous Systems main blocks [1]

In general, autonomous systems usually have four main blocks - sensing, perception, planning, and control. As seen in figure 1 the flow of the blocks is such that the environment is first perceived by the sensors. This information is then converted into meaningful information in the perception block. Based on this information, the planning block decides on a path which the system would need to follow. Finally, the control system of the vehicle executes actions and actuates the vehicle to move according to the path. This approach has been considered in this project too where the deep neural network combines the perception, planning and control blocks in order to output steering angle predictions (thus control the vehicle) by using sensed camera images obtained as input. This results in end-to-end learning that enables a system/vehicle to drive automatically, by learning important details of the path [1].

In this project two different datasets are considered. One is the Yellow track dataset which is short and does not have too many sharp turns or steep terrains. The other is the green track dataset which is comparatively more difficult to train on as the path consists of steep terrains and sharp corners. To learn the paths, the approach is to implement various architectures that have been created since the beginning of 2019. More specifically the results of the yellow track dataset for the models of JNet, JNet_ELU, Pilotnet, and AIRSIM (will be discussed in the later sections) will be analysed. Along with this, the performance of the architectures on the green track dataset will also be investigated.

In the following sections, we go deeper into the related work done, the training methods, the data collected, experiments, comparison of results and the implementation of the autonomous driving feature for all four neural networks.

2 Related Work

Most of the work done in this paper is based on analysing different neural networks on the lightweight Udacity simulator which is available here [2]. This simulator was made open source

on GitHub in January 2019 and from then on, it has proved to be an easy tool to use and test out various architectures. There has been various architectures implemented like the NVIDIA model as in [3], the AlexNet and VGG-Net architectures as in [4]. Another architecture on JNET has been implemented in [5]. This paper was one of the first ones to bring into the idea of making the architecture smaller thereby allowing such a solution to be uploaded in a lightweight manner. Apart from using the UdaCity simulator, there are also other platforms on which architectures have been implemented. One such simulator is AIRSIM, which is developed by microsoft [6]. The first architecture which is also referred to as the hello world program of this platform and which we will refer to as AIRSIM model.

In this project, the work done is to reproduce the results of the NVIDIA (also called PilotNet) and the JNET [5] architectures. Instead of using an ALexNet based method as presented in the paper, the AIRSIM model has been used instead. The original JNET model as presented in the paper has also been modified and compared with here. Following this a qualitative and quantitative analysis of the performance has been made for all the 4 models on a flat terrain and hilly terrain dataset. The comparison metrics and training methods used are similar to those presented in the paper and while the paper and most of the resources online test their solutions on the flat terrain track, this project aims at extending the training and testing to the more difficult hilly terrain.

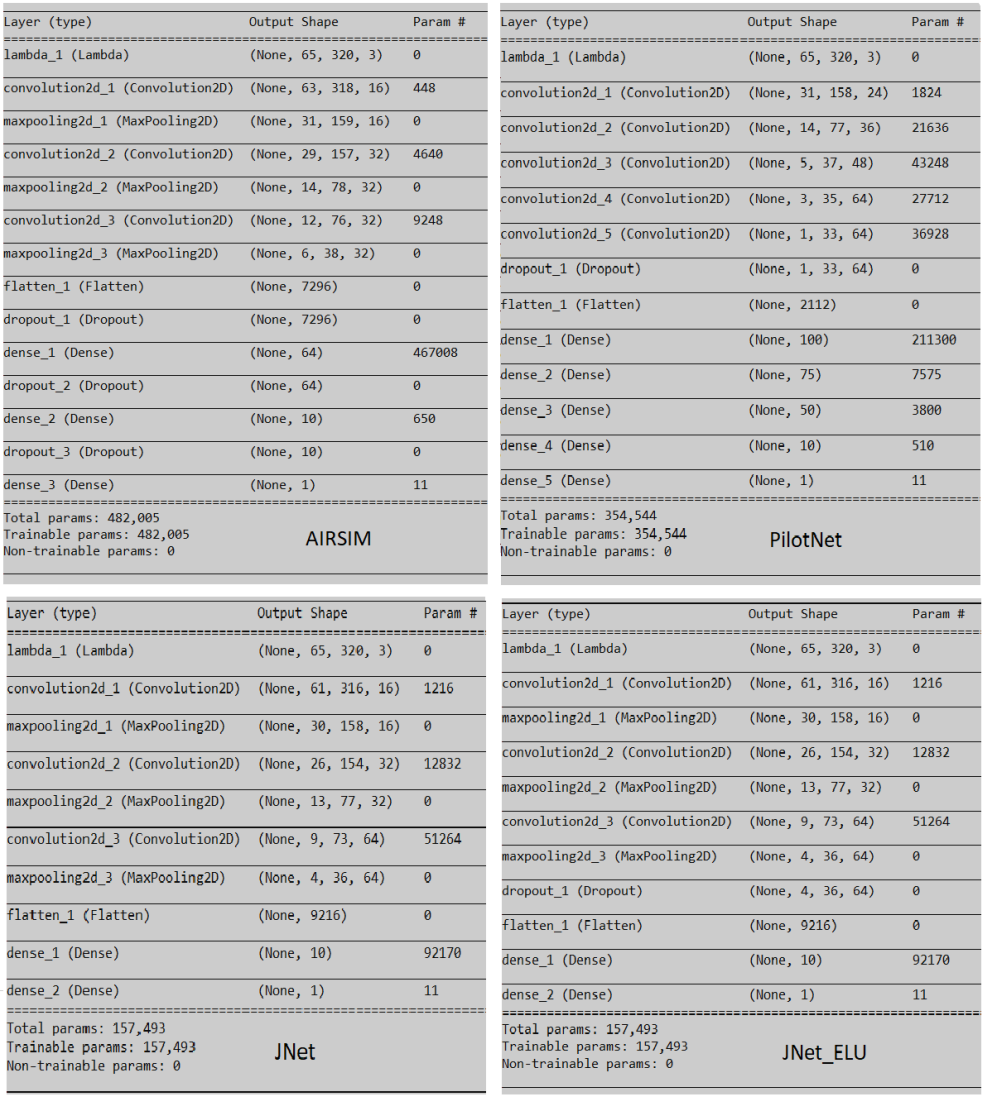
3 Methods

The essential questions that the experiments aim to answer are:-

1. Can the size of the model used in autonomous driving be reduced?
2. Can the reduced model perform well on difficult tracks?

To answer the above questions, we first selected 4 different architectures, based on the number of trainable parameters. We trained these models on the yellow dataset having the same training hyper-parameters (EPOCH, batches and learning rate) and compared the performance to investigate if lighter models would give a similar or even better performance than large sized models. Therefore, by doing this the first question is answered. Following this, we trained the architectures on the more difficult green dataset and noted down its performance. A larger model has a higher capability to learn more complex features of the images and therefore can perhaps track the path better for the same input images. Keeping this in mind we moved onto investigating if lighter models too can learn to traverse more difficult paths thereby helping us to investigate the second question. The architectures chosen were:-

1. AIRSIM
2. PilotNet
3. J-Net
4. J-Net_ELU



AIRSIM

Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 65, 320, 3)	0
convolution2d_1 (Convolution2D)	(None, 63, 318, 16)	448
maxpooling2d_1 (MaxPooling2D)	(None, 31, 159, 16)	0
convolution2d_2 (Convolution2D)	(None, 29, 157, 32)	4640
maxpooling2d_2 (MaxPooling2D)	(None, 14, 78, 32)	0
convolution2d_3 (Convolution2D)	(None, 12, 76, 32)	9248
maxpooling2d_3 (MaxPooling2D)	(None, 6, 38, 32)	0
flatten_1 (Flatten)	(None, 7296)	0
dropout_1 (Dropout)	(None, 7296)	0
dense_1 (Dense)	(None, 64)	467008
dropout_2 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 10)	650
dropout_3 (Dropout)	(None, 10)	0
dense_3 (Dense)	(None, 1)	11

Total params: 482,005
Trainable params: 482,005
Non-trainable params: 0

PilotNet

Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 65, 320, 3)	0
convolution2d_1 (Convolution2D)	(None, 31, 158, 24)	1824
convolution2d_2 (Convolution2D)	(None, 14, 77, 36)	21636
convolution2d_3 (Convolution2D)	(None, 5, 37, 48)	43248
convolution2d_4 (Convolution2D)	(None, 3, 35, 64)	27712
convolution2d_5 (Convolution2D)	(None, 1, 33, 64)	36928
dropout_1 (Dropout)	(None, 1, 33, 64)	0
flatten_1 (Flatten)	(None, 2112)	0
dense_1 (Dense)	(None, 100)	211300
dense_2 (Dense)	(None, 75)	7575
dense_3 (Dense)	(None, 50)	3800
dense_4 (Dense)	(None, 10)	510
dense_5 (Dense)	(None, 1)	11

Total params: 354,544
Trainable params: 354,544
Non-trainable params: 0

JNet

Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 65, 320, 3)	0
convolution2d_1 (Convolution2D)	(None, 61, 316, 16)	1216
maxpooling2d_1 (MaxPooling2D)	(None, 30, 158, 16)	0
convolution2d_2 (Convolution2D)	(None, 26, 154, 32)	12832
maxpooling2d_2 (MaxPooling2D)	(None, 13, 77, 32)	0
convolution2d_3 (Convolution2D)	(None, 9, 73, 64)	51264
maxpooling2d_3 (MaxPooling2D)	(None, 4, 36, 64)	0
flatten_1 (Flatten)	(None, 9216)	0
dense_1 (Dense)	(None, 10)	92170
dense_2 (Dense)	(None, 1)	11

Total params: 157,493
Trainable params: 157,493
Non-trainable params: 0

JNet_ELU

Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 65, 320, 3)	0
convolution2d_1 (Convolution2D)	(None, 61, 316, 16)	1216
maxpooling2d_1 (MaxPooling2D)	(None, 30, 158, 16)	0
convolution2d_2 (Convolution2D)	(None, 26, 154, 32)	12832
maxpooling2d_2 (MaxPooling2D)	(None, 13, 77, 32)	0
convolution2d_3 (Convolution2D)	(None, 9, 73, 64)	51264
maxpooling2d_3 (MaxPooling2D)	(None, 4, 36, 64)	0
dropout_1 (Dropout)	(None, 4, 36, 64)	0
flatten_1 (Flatten)	(None, 9216)	0
dense_1 (Dense)	(None, 10)	92170
dense_2 (Dense)	(None, 1)	11

Total params: 157,493
Trainable params: 157,493
Non-trainable params: 0

Figure 2: Architectures of the four models

In the above, AIRSIM, PilotNet and JNet are directly taken from previous works. In our project, we modified the JNet architecture to have an ELU instead of ReLU activation and a dropout layer. Its simple to remember that JNet_ELU = JNet - ReLU + ELU. The choice of using ELU was because it can produces negative output values which are present in the steering values, which cannot be done by ReLU. The reason for using dropout is to avoid overfitting. Now, Consider the number of parameters from each architecture as in figure 2. We observe that each model has parameters in the order of 10^5. These parameters are directly proportional to the computations performed. Thus, the computations are different between the case of JNET_ELU and PilotNet. For each architecture, the following table presents the number parameters, type of layers and the size of the ".h5" model file (which will be explained later) generated. It can be seen that JNET_ELU has 157k trainable parameters as shown above. This network has nearly half

Layers and Parameters	AlRSIM	PilotNet	JNet	JNet_ELU
Convolution layers	3	5	3	3
Max Pooling	3	1	3	3
Flatten	1	1	1	1
Fully connected layers	3	4	2	2
Dropout Layer	3	1	0	1
Number of trainable parameters	482,005	354,544	157,493	157,493
h5 model size generated	5.55MB	4.10MB	1.83MB	1.83MB

Table 1: Architecture numeric comparison

the number of trainable parameters compared to the PilotNet having nearly 354K parameters.

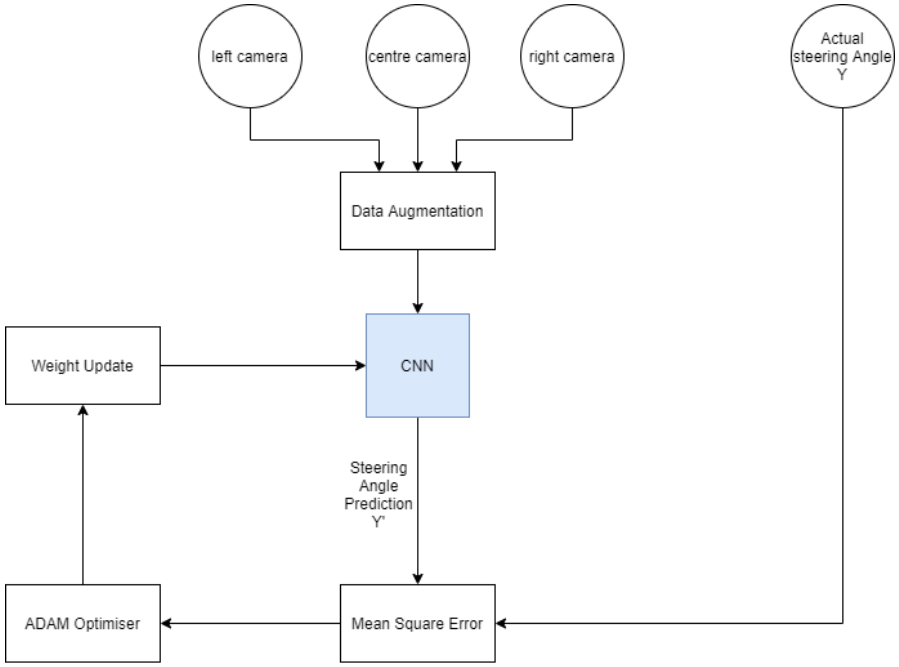


Figure 3: Training for parameter learning [5]

The idea used in the training is as described in the figure 3 and has been taken from the paper [5] where we first obtain the left, center and right camera data and the corresponding steering angle values. The data that is obtained is first passed through a data augmentation step following which it is used to train a CNN. The training method used in the CNN was to use 80% of the obtained data for training and 20% for validation. The training method compares the predicted output to the ground truth (Actual steering angle Y) using a mean squared error function, following which an adam optimiser is used to update the weights of the CNN. The choice of using ADAM was because it is computationally efficient, suited for problems with a large amount of data (like this one) and also combines the benefit of Adaptive gradient descent algorithm and RMS Propagation. More information on ADAM is explained by Jason Brownlee [7]. At the end of the training we observe that by making the CNN learn the weight parameter,

we obtain a controller that readily makes steering angle predictions based on the input data from the camera.

3.1 Dataset

The dataset were the images of the tracks taken by the left, center and right cameras. The simulator that we used for this was the Udacity's Self-Driving Car Simulator. There are 2 tracks that are present in this simulator, the yellow track and the green track.

Figure 4: Yellow Track

Figure 5: Green track

It can be seen from the above images that the yellow track is flat and has very few sharp turns. This is in contrast to the green track which has hilly terrain and many sharp turns. Due to this, we refer to the yellow track as the easy one and the green track as the difficult one. As mentioned in the introduction, the original paper referred to has made a comparative study on only the yellow dataset. In our study, we have done the same for the green dataset and have drawn conclusions based on a comparison of the models on both the datasets.

As seen from the figure 3, before feeding the data into the neural network, there is a data augmentation block. The block is required to convert the data into a size that is compatible

with the architecture and also that consists of pixel values that are relevant to the steering angle prediction. The process that is followed in the original paper [5] is to convert an input image of size 320*160*3 into a cropped up image of size 320*65*3 such that the portions of the front car bonnet and the sky are cropped out, as shown in figure 6.

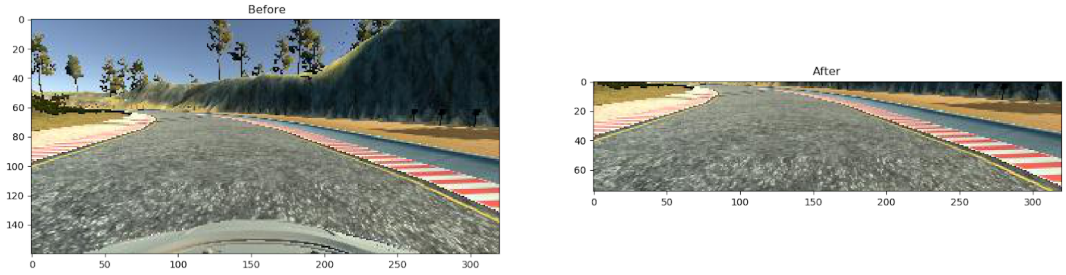


Figure 6: Cropped input image

While the above works for straightforward tracks, in the case of green tracks the situation is more different primarily because there are many moments when the car is not horizontal and the path in front of it lies at the upper region of the camera image. This can be seen in one such part of the map as shown in the front camera image.

Figure 7: Car front view

In order to deal with such scenarios, the data sets were only cropped in order to remove the car bonnet pixel data, thus resulting in the figure 8.

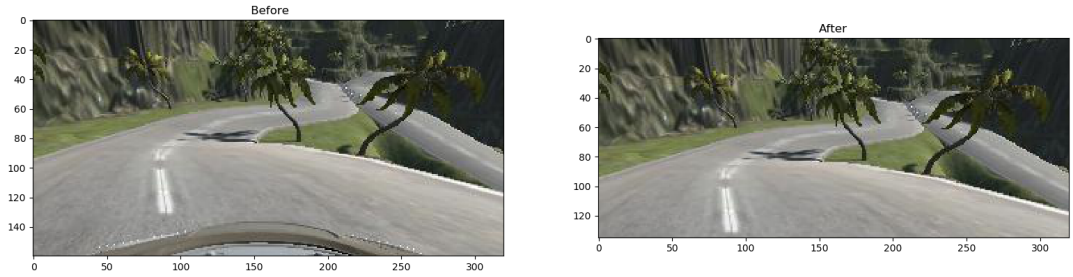


Figure 8: Car bonnet removed

Further, the images were then resized into the size of 320*65*3, as per the requirements of the architecture in the paper. We also note that this dimension worked well for the AIRSIM model that was taken from another source. This type of data augmentation is also used for the yellow dataset, and it did not hamper the final functionality of predicting reasonable steering angle values (as will be seen later).

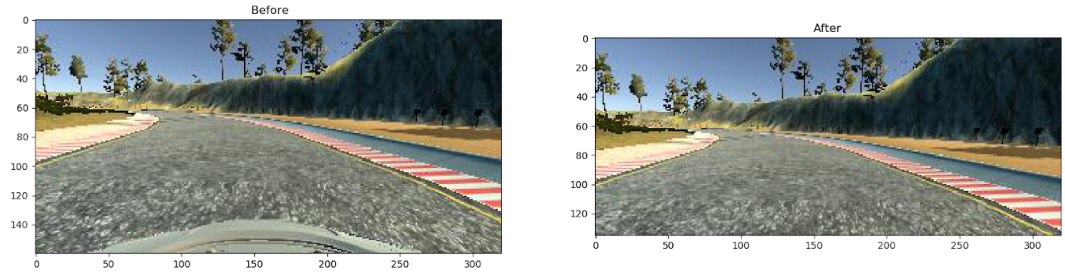


Figure 9: Resized image

3.2 Training Algorithm

Along with the training images, the corresponding values of steering angles, throttle, reverse and speed were obtained. All this was automatically generated into an excel sheet format by the simulator, as can be seen in the figure 10.

1	CENTER	LEFT	RIGHT	STEERING	THROTTLE	REVERSE	SPEED
53	C:\Users\\	C:\Users\\	C:\Users\\	0.05	1	0	30.19002
54	C:\Users\\	C:\Users\\	C:\Users\\	0.2	1	0	30.18184
55	C:\Users\\	C:\Users\\	C:\Users\\	0.4	1	0	30.15928
56	C:\Users\\	C:\Users\\	C:\Users\\	0.55	1	0	30.1405
57	C:\Users\\	C:\Users\\	C:\Users\\	0	1	0	30.18647
58	C:\Users\\	C:\Users\\	C:\Users\\	0	1	0	30.18961
59	C:\Users\\	C:\Users\\	C:\Users\\	0.15	1	0	29.8988
60	C:\Users\\	C:\Users\\	C:\Users\\	0.35	1	0	30.17633
61	C:\Users\\	C:\Users\\	C:\Users\\	0.6	1	0	28.99148
62	C:\Users\\	C:\Users\\	C:\Users\\	0	1	0	28.46664
63	C:\Users\\	C:\Users\\	C:\Users\\	0	1	0	28.85051
64	C:\Users\\	C:\Users\\	C:\Users\\	0	1	0	28.56289
65	C:\Users\\	C:\Users\\	C:\Users\\	0	0.9395212	0	28.55033

Figure 10: Excel sheet containing the the data information

For our experiments, we used the CENTER, LEFT, RIGHT as the training data and only the steering values for the output value. It is important to note that in a forward pass of training only one amongst the 3 images is used. Initially the strategy was that we used all the training fields across the entire rows and then trained the model. However there were 3 main demerits with this way for training:-

1. The time taken to load all the images into the code even before training could begin was very high. An example of this is that for the green dataset where 58K image data points were present, it took 10-15 minutes to load the data itself.

2. Using the left and right training images was not that necessary for the driving because in the final working only CENTER images were to be taken. The LEFT and RIGHT images were only used to help improve the steering by ensuring that such difficult steering scenarios are possible to maneuver.

3. When we compared the final values of training and validation loss with another method (which will be mentioned below) that instantly began the training instead of loading the data, we found that the performance was very similar. Therefore the additional time taken would be unnecessary in the long term goal of predicting the correct steering value.

3.3 Modified training method

The method explained here has been originally created by Naokishibuya [8]. In this project, we have modified this code as per our requirements. The fundamental aspect of the algorithm used in that it does not train over all the images in an EPOCH. Rather there is a parameter called samples per EPOCH that can be used to determine the number of samples to be considered per training EPOCH. Further, mini-batch gradient descent was used where for each gradient descent step, batch_size value of images was taken. The selection of such images was done by generating a random number and comparing it with a selected threshold (In our case 0.75). If the number was lesser than this then LEFT or RIGHT images would be taken, else only CENTER camera images. Once the image was selected, it would be sent to the data augmentation step (explained earlier) following which the images (of size batch_size) and its corresponding steering angle values

Convolution layers	**value**
Yellow track Training EPOCHS	10
Yellow track Samples/EPOCH	10100
Green track Training EPOCHS	20
Green track Samples/EPOCH	46500
Learning Rate	10^{-4}
Mini batch number	100

Table 2: Hyper-parameters used

would be sent for training.

Upon trying different values, the best values were obtained by setting batch_size = 100, EPOCH=10 and samples_per_EPOCH = 10100 for the yellow dataset and samples_per_EPOCH = 46500 and EPOCH=20 for the green dataset. The samples_per_EPOCH was chosen based on the amount of data that was used to train the model. For both the yellow and green datasets, we ran the car for 10 laps to obtain the required data as in figure 10. In the case of the yellow dataset, we obtained around 30K data points and setting the training to only train on 10100 samples per epoch with 10 EPOCH values was found to work OK. However, while using the green dataset the training example was 46K and therefore we decided to use more training samples corresponding to this number and for a higher value of EPOCH = 20. A higher value of batch size of 100 was taken because of higher convergence rate to the optimum value of weights. We ensured that it did not go beyond this value as this would possibly lead to overshooting the optimal value of gradient. A learning rate of 10^{-4} was chosen and found to work okay for most of for architectures in the yellow dataset.

After every EPOCH, the code generated a ".h5" file based on a lower value of val_loss produced by the training. A ".h5" file is basically a hierarchical data format file that consists of the model in a binary format and is used to drive the car in the simulator. More on .h5 files can be found here [9]. It is important to note that while we trained the data set such that a ".h5" was generated only when a better val_loss is obtained, we later realised that many times the models giving a lower training value proved to give better steering value. Due to this, we set the training such that it would generate a model after every EPOCH was complete. Thus, we obtained 10 ".h5" model files when EPOCH=10. The summarise values of hyper-parameters used have been mentioned in table 2.

4 Results

After the training of the networks was complete a quantitative analysis of the loss value and the steering angles predicted as well as a qualitative analysis of the nature of successful driving on the representative track in the autonomous mode was performed. This type of analysis was chosen because the loss values would give us the extent to which the model was trained. The predicted steering values are obtained from the trained model, where the model gives out the output based on the image from the front camera. This can be seen from the below figure

4.1 Loss value comparison

Model Name	Training Loss	Validation Loss
Yellow track		
JNet_ELU	0.0248	0.0212
PilotNet	0.0266	0.0208
AIRSIM	0.0260	0.0214
JNet	0.0245	0.0222
Green track		
JNet_ELU	0.0867	0.1105
PilotNet	0.1035	0.1105
AIRSIM	0.1005	0.1062
JNet	0.0787	0.0998

Table 3: Loss value comparison

Figure 11: General Mechanism

We recorded the steering angle information in order to compare the output for different models for the same track. Further, a qualitative analysis on the driving has been done in order to compare the smoothness of the drive when following the road.

The table specifying the loss values and the graphs are presented. It is important to note that the aim in each case is to reduce the mean square error values, which is indicated by the training and validation losses. We make the following observations from the table 3 and figures 12 and figures 13:

1. The training and validation loss for green track dataset is higher than the corresponding losses of yellow track for all the models. This can be attributed to the level of complexity in the track consisting of sharp turns and also of adjacent paths which can be challenging for the model to learn.

2. In the case of the yellow track, the losses for the 5.55MB sized AIRSIM model is higher than JNet_ELU. This is possibly due to the requirement for more training EPOCHS for the AIRSIM, which has more parameters to train.

3. In the case of the yellow track, if we consider the values of the losses to the 2nd decimal place we find that the losses are similar.

4. In the case of the yellow track, 10 EPOCHS is enough for enabling a smooth drive in all models (as will be seen) but it is not enough for asymptotic convergence of the loss values.

5. In the case of the green track, JNet_ELU and JNet appear to give lower values of training and validation loss when compared to PilotNet and AIRSIM.

6. There is a downward trend of the loss for each model, which indicates that the model is learning the parameters and not stuck at a constant value.

Considering the observations, particularly 2) and 4) we can infer that using a lighter JNet architecture and JNet_ELU, we can get similar and even better loss values for a straightforward as well as difficult track.

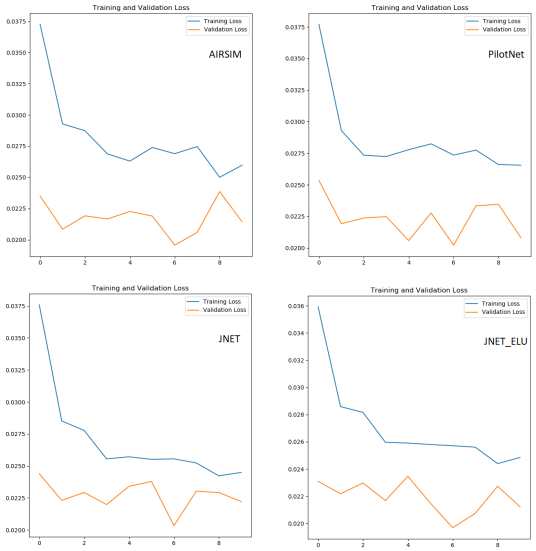


Figure 12: Yellow track results

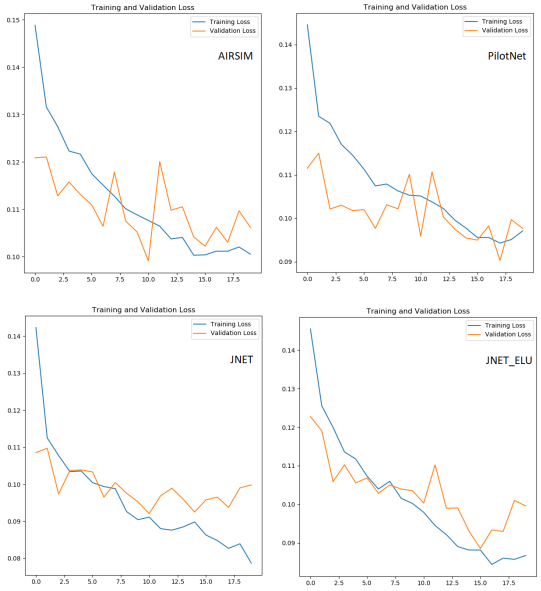


Figure 13: Green track results

4.2 Driving Quality comparison

To compare the driving quality or the behaviour of the trained model on a real scenario where the agent is left to drive on its own, we first focus on the models ability to complete 1 lap which can take either yes or no values. When the model is able to complete 1 lap, we move onto determine the curve handling capability and keeping to the middle of the road, or the centre trajectory as it is called in the analysis. These factors can take either good, medium and bad parameter values. Thus by doing this, we compare the models based on its capability to complete the track in a satisfactory manner for 1 lap, which is also representative of the fact that it can complete the track for more laps as has been done in [5].

4.2.1 Yellow track

	AIRSIM	PilotNet	Jnet	Jnet_ELU
Completion of 1 lap	yes	yes	yes	yes
Curve Handling	good	medium	good	good
Keeping Center trajectory	good	medium	medium	medium

Figure 14: Yellow track driving quality

We make the following observations:-

1. All the tracks were able to complete the lap after being trained for only 10 EPOCHS.
2. The curve handling of the most models was better when compared to keeping the center trajectory.
3. AIRSIM gave the best performance amongst all the models. This is probably due to the high number of parameters
4. JNet and JNet_ELU gave similiar driving behaviour.

4.2.2 Green track

	AIRSIM	PilotNet	Jnet	Jnet_ELU
Completion of 1 lap	no	no	yes	yes
Curve Handling	-	-	good	good
Keeping Center trajectory	-	-	good	medium

Figure 15: Green track driving quality []

We make the following observations:-

1. For a training of 10 EPOCHS and a learning rate of 10^{-4}, only JNet and JNet_ELU were able to complete the track for 1 lap.
2. JNet was able to keep to center of the trajectory better as compared to JNet_ELU.

(a) Case 1 (b) Case 2

Figure 16: Green track error points

Further, it was observed that AIRSIM and PilotNet model agents would get stuck and not run at 2 points of the track as shown in figures 16a and 16b. In these cases, we find that the common element is that there are 2 paths that are sensed by the CENTER camera. Due to this we infer that possibly the agent is not able to predict the right steering angle value. To tackle this occurence, we increased the number of EPOCHs to 50, however this did not improve the scenario and required the learning rate to be tuned. However as these 2 models were larger and could not be trained for the hyper-parameters of table 2, we decided to continue with the analysis only for JNet and JNet_ELU.

4.3 Steering value comparison

To quantitatively compare the steering value predictions of the 4 models, we adopt 2 separate methods for yellow and green track respectively. This has been done due to the nature of the tracks and will be explained further below. It is important to note that a negative value of angle corresponds to steering to the left (and therefore positive for right).

4.3.1 Yellow track

The comparison methodology used here relies on the fact that there are more instances of having a straight path in the yellow track and the car spends most of the time in a state where its steering value needs to be 0 or almost 0. Considering this idea, the histogram of the predicted steering values for each model has been plotted. We make the following observations:-

1. In all cases, most of the predicted steering values lie in the centre of the approximately -2 to +1 region which indicates that all the 4 models were able to traverse the straight paths for a significant amount of time.

2. The steering values produced are in the range of -25 to 21 in the case of PilotNet, which is higher than the case of the other models which roughly lie in the range of -17 to 18. This also indicates that 10 EPOCHS is not enough to train the PilotNet and certainly more epochs are needed for a smooth journey.

3. The steering angle values and the numbers of such angles produced by JNET is similar to AIRSIM, which has more trainable parameters. This implies the functionality of traversing a flat track can be done by the lighter model.

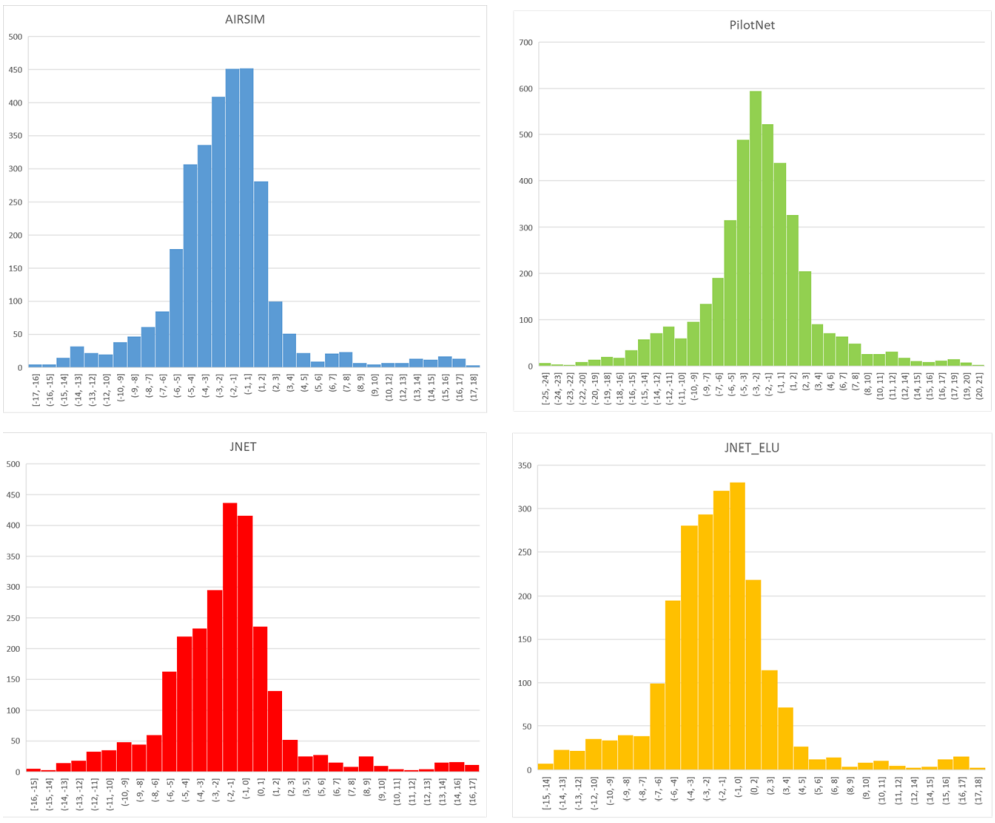


Figure 17: Yellow track predicted steering values

4.3.2 Green track

As the track had sharper turns and did not require the agent to maintain a zero prediction angle for majority of the time, therefore the previous quantitative analysis could not be performed. Instead, the generated predicted steering angle values for 1 lap was plotted as a function of when they arrive. This meant that the x-axis represents the progress of time. The result was only plotted for JNet and JNet_ELU as these models were able to complete the track based on the hyper-parameters mentioned in table 2.

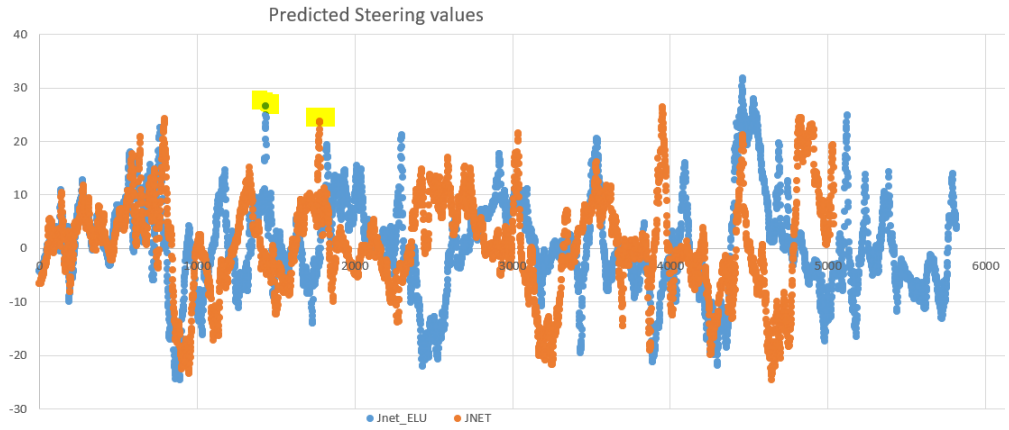


Figure 18: Green track predicted steering values

The resultant plot is shown on figure 18 and we make the following observations:-

1. The behaviour of the models is similiar for the first 1000 values generated.
2. If we observe the highlighted values carefully, it is observed that in the case of JNet_ELU, similiar values of predicted steering occur before the JNet value.

The above implies that in the case of JNet, more steering prediction values are generated as compared to JNet_ELU. This helps JNet stick to the trajectory more and make it complete the track faster which can be seen in the graph, where more JNet_ELU prediction values are necessary to complete 1 lap.

4.4 Inferences

From the qualitative and quantitative study (loss values and predicted steering values) of the models we make the following inferences:-

1. 10 EPOCH is sufficient for all the models to learn simple tracks, like the yellow one. While 10 EPOCH's works for lighter model, on more complicated tracks a higher number of EPOCHS or higher learning rate is required for larger models.
2. If space is the constraint, then models of a lesser size (JNet and JNet_ELU) can be used instead of (AIRSIM and PilotNet) for simple as well as complicated tracks.
3. JNet gives a smoother and more controlled trajectory than JNet_ELU in terms of curve handling and keeping center trajectory. Further, it also appears to complete the lap faster.

5 Ablation Study

In order to get an in depth understanding of the possible improvements of modifying the JNet model, we include this ablation study which compares the performance of 4 variants of the Jnet model on the same dataset having the same hyper-parameters values of number of epochs, samples per epoch and the batch size during training. This study was conducted by using the yellow

Model variant	Training loss	Validation loss	Training accuracy	Validation accuracy
JNet	0.0189	0.0126	0.3128	0.7897
JNet + dropout	0.0197	0.0126	0.3129	0.7885
JNet + ELU	0.0207	0.0132	0.3130	0.7897
JNet_ELU	0.0219	0.0128	0.3125	0.7885

Table 4: Ablation study of JNet

track dataset and having the batch size as 40, the samples per epoch as 12000, the learning rate as 10^{-4} and the number of epochs for training as 10. It is important to recall that while the original JNet architecture consists of the ReLU activation with no dropout, the modified JNet_ELU is basically JNet + dropout + ELU activation function as can be seen from figure 2.

From the table 4 and figure 19, we observe that:-

1. There is a downward trend in the loss curves, which indicates learning taking place.
2. JNet_ELU has a very similar result to the JNet architecture with the relu activation.
3. As can be seen in JNet_ELU v/s JNet+ELU and JNet v/s JNet+dropout, when dropout is used the training loss increases, which indicates that the dropout is indeed preventing overfitting.
4. The validation and training accuracy values are almost similiar for each case

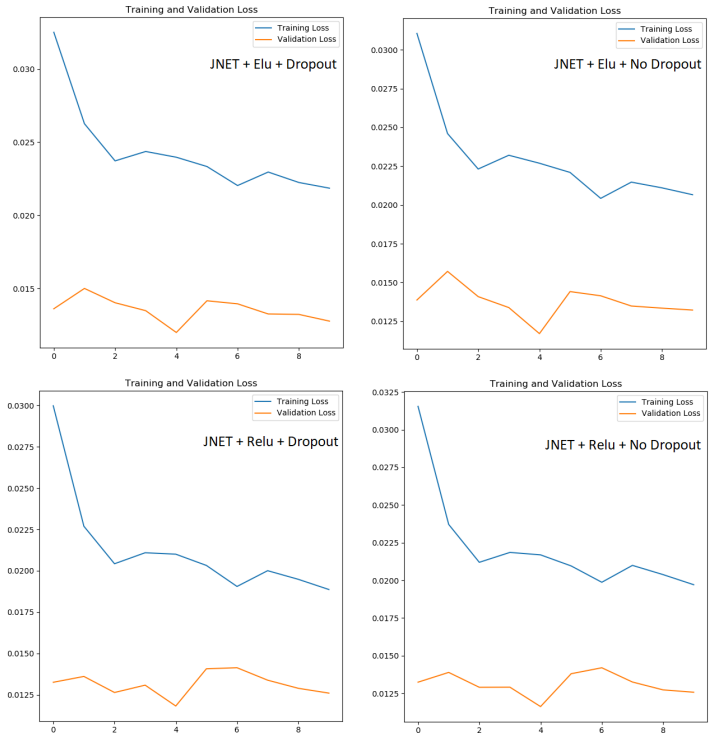


Figure 19: Ablation study graphs

From the above results, we get that the performance of the JNet_ELU is very similar to the JNet, with the dropout. It has a slight increase in training loss but shows a very similar validation loss and performs well in the autonomous mode in the simulator. Moreover the JNet model has a training a lesser training loss when compared to JNet_ELU. This is possibly the reason why is results in a better driving experience for the green dataset as was seen in figure 15.

6 Shortcomings

In our implementation of the models, we noticed some shortcomings which can possibly be tackled as a further research study. These are:-

1. Only steering value has been used as the output predicted value. As seen in figure 10, a collection is Steering, throttle, reverse and speed is obtained along with the images. Thus, instead of only using the values of steering, a weighted combination can be used in order to get lower loss values.

2. The controller designed through this method can only be used in applications where the path is predefined and known. Applications like warehouse robots and cleaning robots that follow a set trajectory are examples of this. For scenarios where the agent is required to learn the environment on its own, algorithms based on Deep Reinforcement learning or even self-supervised learning can be used.

3. In order to deploy actual autonomous vehicles into the field, this type of controller is only one part of the picture and needs to work in conjunction with other algorithms like image segmentation and object detection in order to detect pedestrians and other obstacles.

4. The code used to implement this model is quite old. In fact, for the deployment of files onto embedded devices, ".h5" files is not as popular as using frameworks like TensorFlow Lite. Updating the code from Keras to TensorFlow 2 and Pytorch would be needed in the future.

5. As deployment in resource constraint environments is the aim, the use of Binary Neural Networks can possibly be used in this application. This is quite possible as BNN's have been used for developing size constraint sensors previously, like Ceva SensPro.

7 Conclusion

The JNet model and its modification JNet_ELU of size 1.83MB has been proved to be a possible solution for low resourced hardware platforms in order to give perform autonomous driving over easy and difficult tracks. Moreover, when compared to models of higher complexity such as PilotNet (4.10MB) and AIRSIM (5.55MB), it achieves similar performance (and even better as seen for the difficult dataset) when trained for low EPOCH values of 10 to 20. Thus, we were able to get very similar training loss and driving results of [5] and extend it further to make it work on a harder scenario (green dataset).

Moreover, the project also introduces an easy to use training simulator that can be installed in order to test various models. The link to this installer and all the important code and results can be found in this GitHub Link. Further, we recorded the driving of the working models on the yellow and green dataset and these can be found in table 5.

Description	**Link**
JNet_ELU Green Dataset	Link
JNet Green Dataset	Link
JNet_ELU Yellow Dataset	Link
JNet Yellow Dataset	Link
AIRSIM Yellow Dataset	Link
PilotNet Yellow Dataset	Link

Table 5: Links to video recording

References

[1] 7 key challenges impacting the autonomous vehicles.

[2] Udacity. udacity/self-driving-car-sim.

[3] Mariusz Bojarski, Mariusz Bojarski, Ben Firner, Beat Flepp, Larry Jackel, Urs Muller, Karol Zieba, and Davide Del Testa. End-to-end deep learning for self-driving cars, Apr 2018.

[4] Aditya Babhulkar. Self-driving car using udacity's car simulator environment and trained by deep neural networks, 2018.

[5] Jelena Kocić, Nenad Jovičić, and Vujo Drndarević. An end-to-end deep neural network for autonomous driving designed for embedded automotive platforms. 19:2064, 2019.

[6] Microsoft. microsoft/autonomousdrivingcookbook.

[7] Jason Brownlee. Gentle introduction to the adam optimization algorithm for deep learning, Nov 2019.

[8] Naokishibuya - overview. `https://github.com/naokishibuya`.

[9] Introduction to hdf5. `https://support.hdfgroup.org/HDF5/doc/H5.intro.html`.