Behavioural Cloning for Autonomous Vehicles

Vatsal Sanghavi

Bhavin Kothari

Viraj Shah

BUDT758B: Big Data & Artificial Intelligence

Dr. Kunpeng Zhang

December 18, 2020

Table Of Contents

Introduction **2**

Our Solution **4**

Preliminary Attempt: Modifying comma.ai’s model 4

Final Attempt & Model Selection: Modifying NVIDIA’s model 5

Training and Hyperparameter Tuning 6

Testing Phase 6

Data Description & Characteristics **7**

Data Preprocessing Pipeline 7

Dataset Input 7

Data Balancing 8

Data Augmentation 9

References **15**

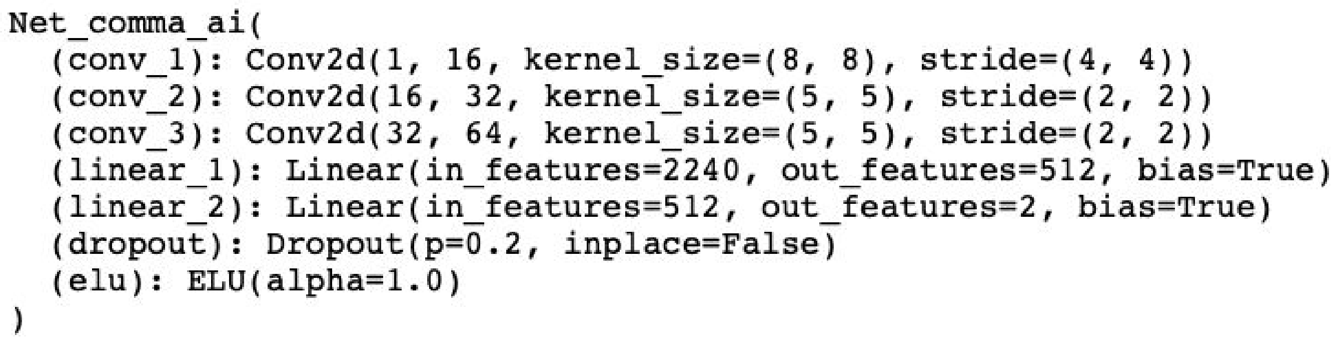
Introduction

This project tries, in essence, to aid solving the challenges of autonomous lateral control by leveraging deep learning enabled behavioural cloning. The scope of this project is limited to training a neural network for steering a car in a simulated environment – we used a video game, Grand Theft Auto-V (GTA-V), for this purpose – by predicting correct steering angles while driving autonomously. Self-driving technology is the next big frontier waiting to be conquered in the ongoing technological revolution. Massive tech companies including the likes of Apple, Google, Tesla, and Facebook are trying to refine and perfect self-driving algorithms by leveraging the power of Artificial Intelligence and Big Data. Autonomous driving is one of the most challenging problems of our times, which has acted as a primary stimulus for us in undertaking this project. We intend to train a model which is capable of autonomously driving a car within a simulated environment. Our training data will comprise of screen captures generated while driving on the roads of GTA-V, along with corresponding steering angles and throttle values. Each of these frames will then be passed as an input to a deep neural network, which will in turn, learn the responses. The model is evaluated on new unknown scenarios wherein it will try and predict the responses with the help of trained parameters (Owoyemi, 2017). One major factor – which is conducive for deep learning algorithms, in general – aiding the performance of this model is the enormous data upon which it has been trained. We also noticed that GTA-V's simulated tracks provide a real-time driving experience, which very closely mimics humans driving on roads. This specific property will be helpful in translating our model from the virtual environment into the real world, which is, in summary, our intention for this project.

Existing Methods

Comma.ai's open-source steering model (Santana & Hotz, 2016) was considered to be the baseline model for this project. It was successful in handling real-world scenarios. Its architecture was follows:

1) Initial Lambda layer which mean-centers the images.
2) Besides the model's simplistic design, one striking characteristic was the use of ELU as a non-linear activation function instead of ReLU. This was done with a hope of speeding up learning, getting better classification accuracies along with the benefit of easy modeling.
3) No pooling layers were used across the entire architecture; the rationale being that a pooling layer made the output of CNN invariant to shifts in the input to some extent, which was not desirable in our case to keep the vehicle on-track (i.e., center of the road).



```
Net_comma_ai(
  (conv_1): Conv2d(1, 16, kernel_size=(8, 8), stride=(4, 4))
  (conv_2): Conv2d(16, 32, kernel_size=(5, 5), stride=(2, 2))
  (conv_3): Conv2d(32, 64, kernel_size=(5, 5), stride=(2, 2))
  (linear_1): Linear(in_features=2240, out_features=512, bias=True)
  (linear_2): Linear(in_features=512, out_features=2, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
  (elu): ELU(alpha=1.0)
)
```

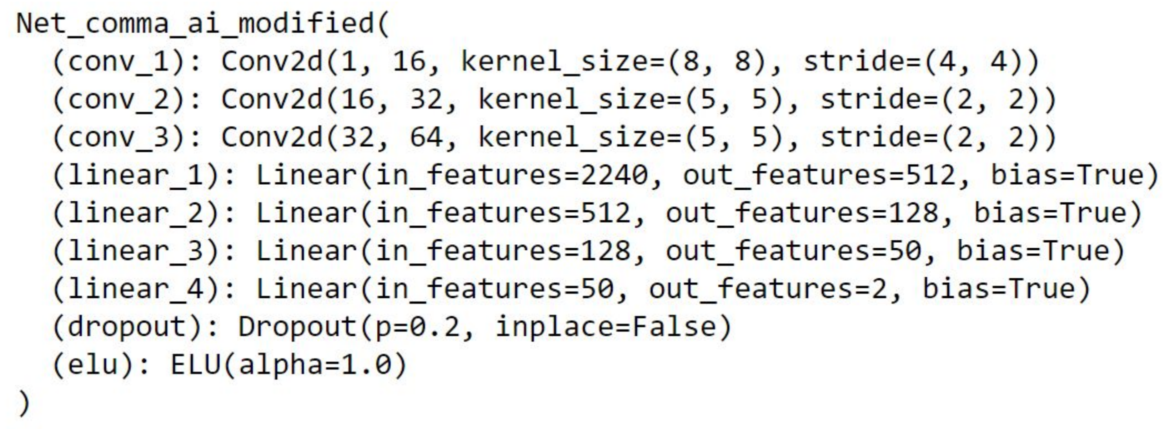
Fig 1: Comma.ai Model Configuration

Our Solution

Preliminary Attempt: Modifying comma.ai's model

Considering the fact that the task of predicting steering angles and throttle values from an input image was a computer vision task, we selected a Convolutional Neural Network (CNN) architecture as our choice of neural network. Pytorch was used for implementing the training and validation processes. This required an additional transformation of converting images into tensor form. This was followed by the creation of a training and validation data loader which configured the data to be suitable for the CNN architecture.

In this project, our very first approach was to implement a modified version of comma.ai's simplistic model - for which the source code on Github was used as reference (Kacmajor, 2018). We settled on implementing a modified CNN model which had the following configuration:



```
Net_comma_ai_modified(
  (conv_1): Conv2d(1, 16, kernel_size=(8, 8), stride=(4, 4))
  (conv_2): Conv2d(16, 32, kernel_size=(5, 5), stride=(2, 2))
  (conv_3): Conv2d(32, 64, kernel_size=(5, 5), stride=(2, 2))
  (linear_1): Linear(in_features=2240, out_features=512, bias=True)
  (linear_2): Linear(in_features=512, out_features=128, bias=True)
  (linear_3): Linear(in_features=128, out_features=50, bias=True)
  (linear_4): Linear(in_features=50, out_features=2, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
  (elu): ELU(alpha=1.0)
)
```

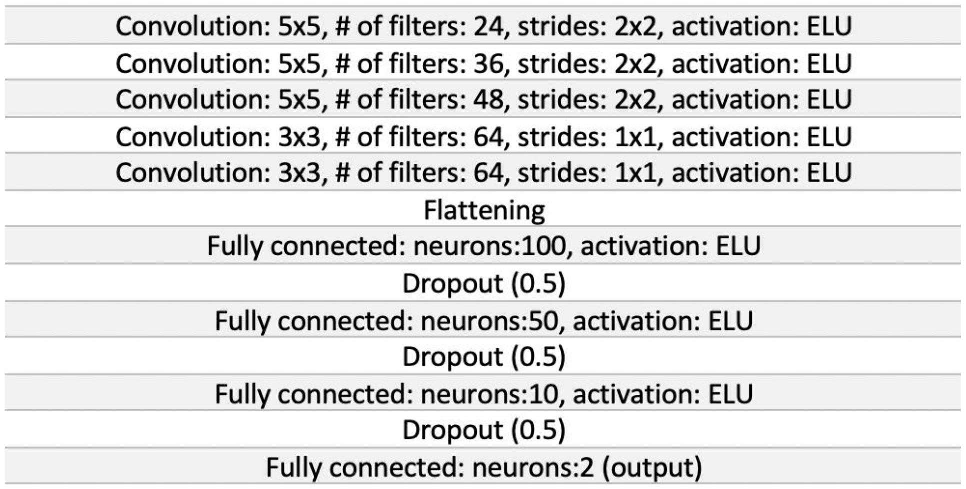
Fig 1: Modified comma.ai Model Configuration

This model had 7 trainable layers and followed standard design practices for CNNs: base convolutional layers' height progressively decreased while the depth increased, while implementing fully connected feedforward layers towards the end.

The Exponential Linear Unit (ELU) activation function was considered to ensure an increased non-linearity. We found that this model had a lower loss on the training dataset but a higher validation loss. Hence, we introduced dropout layers to combat this issue. This model used an Adam optimizer to ensure that the learning rate was automated. Batch size was considered to be 500. We even experimented with the number of epochs and ranged them between 1 and 20.

Final Attempt & Model Selection: Modifying NVIDIA's model

We thought that the modified comma.ai model was suitable for testing over new unknown GTA-V tracks - until we compared its performance to Nvidia's architecture for end-to-end learning for self-driving cars, which is a well-known CNN architecture with 27 million connections and 250 thousand parameters (Bojarski et. al, 2016). The modified CNN architecture that we finally made use of had the following configuration (Kacmajor, 2018):



Convolution: 5x5, # of filters: 24, strides: 2x2, activation: ELU
Convolution: 5x5, # of filters: 36, strides: 2x2, activation: ELU
Convolution: 5x5, # of filters: 48, strides: 2x2, activation: ELU
Convolution: 3x3, # of filters: 64, strides: 1x1, activation: ELU
Convolution: 3x3, # of filters: 64, strides: 1x1, activation: ELU
Flattening
Fully connected: neurons:100, activation: ELU
Dropout (0.5)
Fully connected: neurons:50, activation: ELU
Dropout (0.5)
Fully connected: neurons:10, activation: ELU
Dropout (0.5)
Fully connected: neurons:2 (output)

Fig 2: Final Model - Modified NVIDIA Model Configuration

We understood all the flaws from our previous model and tried to correct them in this final model. We ended up with a final model that consisted of 5 convolution layers followed by 4 fully connected feedforward layers (Kacmajor, 2018).

Training and Hyperparameter Tuning

The first step was to perform a train/validation split. We used 80% of our data for training and the remaining 20% for validation purposes. Adam optimizer was used to minimize the MSE and a learning rate of 0.001 produced quality results even when it was not extensively tuned. Dropout layers with a dropout rate of 0.5 were included to combat the issue of overfitting. We even attempted to add L2 regularization but it had to be excluded from the final model as it gave worse results over the simulated GTA-V test track. Our model had 9 trainable layers and 465816 trainable parameters (excluding biases). We considered 20 epochs with a batch size of 500.

Testing Phase

In the testing phase, our model ran in the background while simultaneously controlling the steering angles and throttle values of the vehicle within the game. This was made possible by implementing the vJoy simulation software (vJoy, 2020) in the background. This autonomous control was made possible by the training that it had received earlier from the images of the game's screen captures. The following footage shows our model's performance while driving autonomously - without any external controls - at night, in the video game, GTA-V. https://github.com/bhavinkothari57/bhavinkothari57-Behavioral_Cloning_For_Autonomous_Vehicles/tree/master/Output%20Video

Data Description & Characteristics

Data Preprocessing Pipeline

The following image is an intuitive way to depict our approach towards data preprocessing and its augmentation. Each step in this pipeline has been explained in detail in the subsequent paragraphs (Owoyemi, 2020).

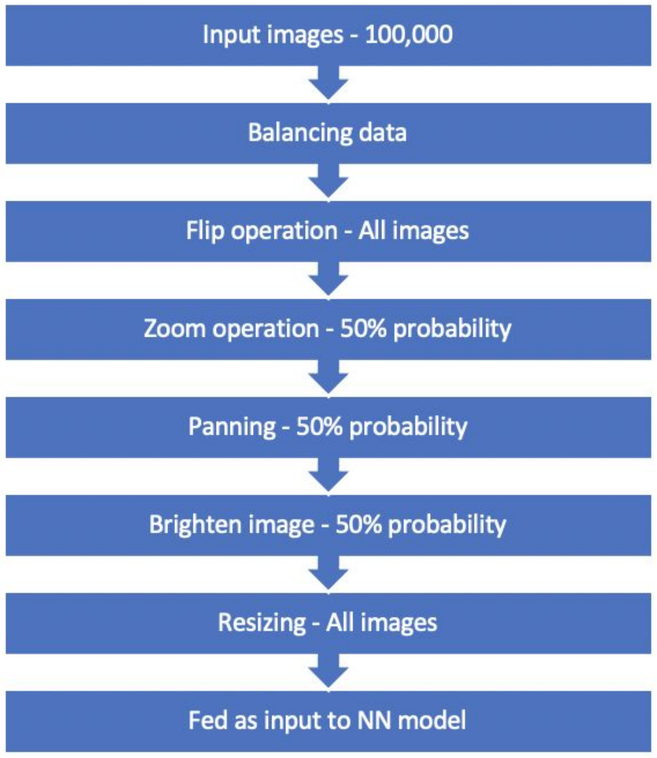

Input images - 100,000
Balancing data
Flip operation - All images
Zoom operation - 50% probability
Panning - 50% probability
Brighten image - 50% probability
Resizing - All images
Fed as input to NN model

Fig 3: Data Preprocessing Pipeline

Dataset Input

We have used 100,000 images with respective steering angles and throttle values. These images have been collected by driving a car on GTA-V's simulated tracks. The training data was collected from an online source to avoid playing GTA-V for a massive number of hours. We obtained our training set from the following source: https://drive.google.com/drive/folders/1R787vkWaMe5nsWyLpbXTG55aUv4Yte (Karovalia, 2020)

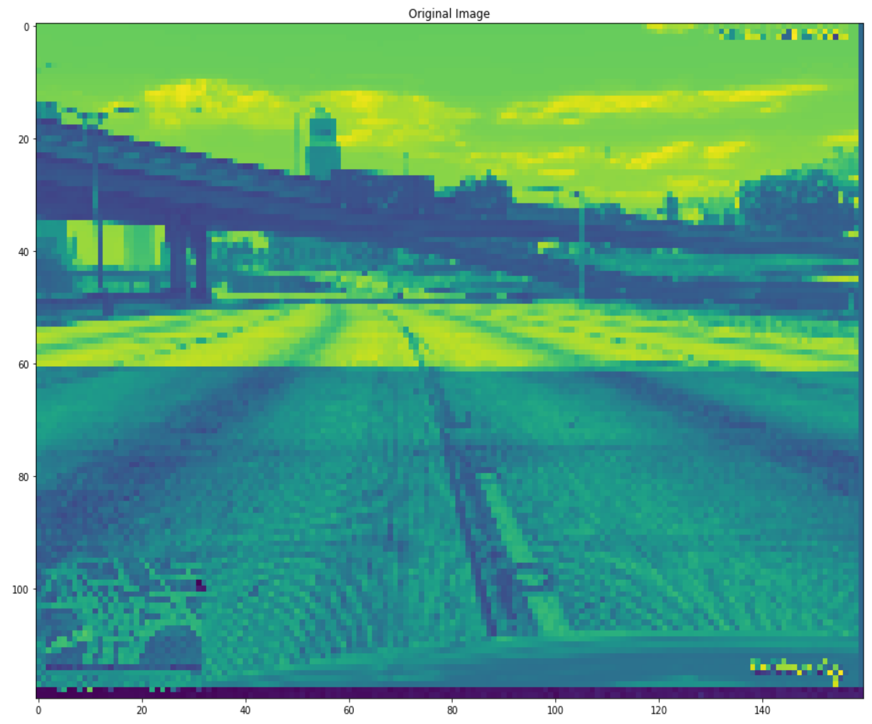


Fig 4: Sample Dataset Image

Data Balancing

This became necessary due to the track's nature, which ensured that most samples had a steering value of 0. To avoid making our neural network model biased towards driving straight, we filtered a random sample of images which had a 0 value for steering angle. This becomes evident from the histogram plots.

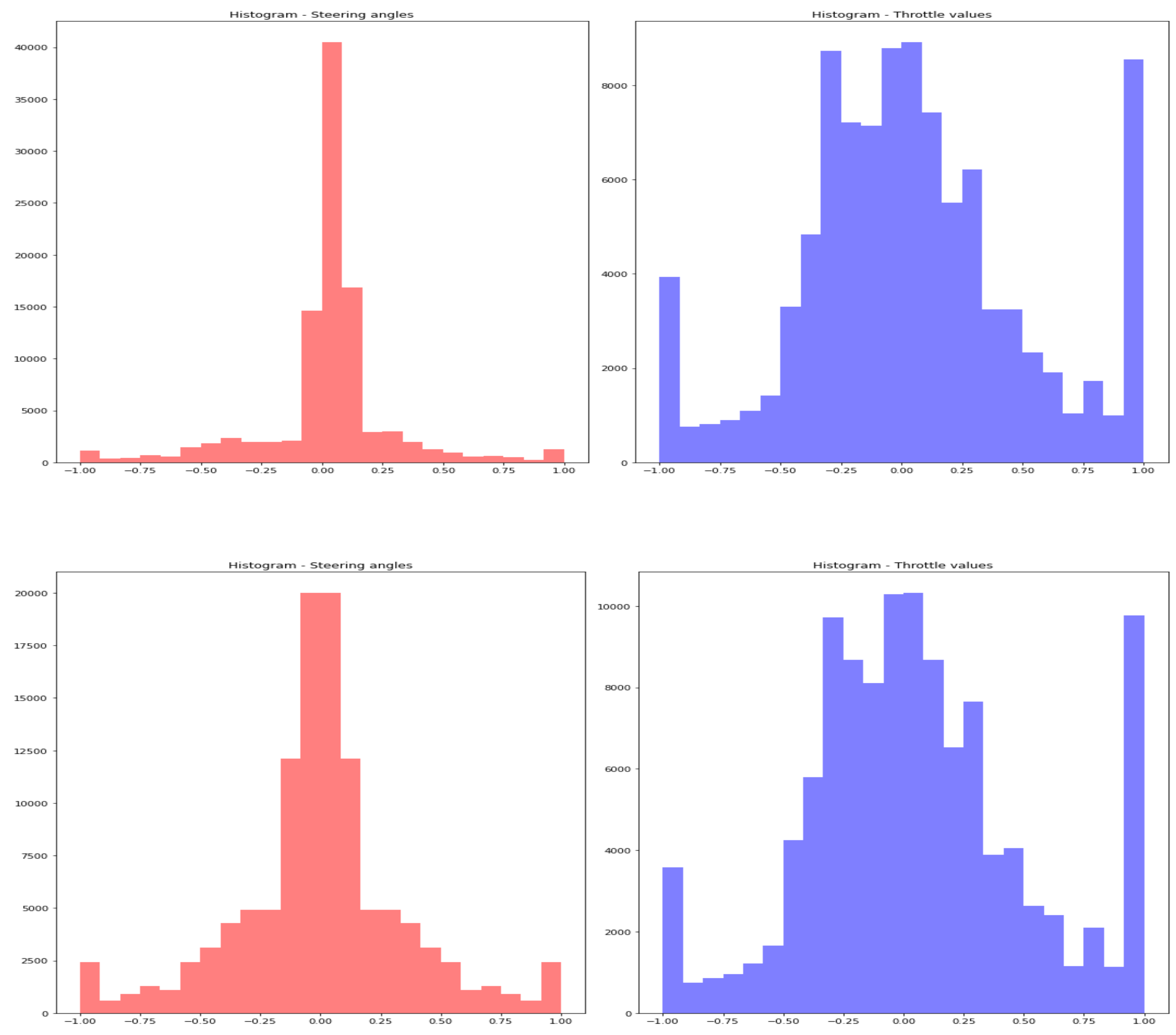


Fig 5: Data Balancing

Data Augmentation

With an aim to increase training data and maintain a balance between left and right steering angle targets, we randomly flipped images and changed signs for the steering angles. This helped in better generalizing our model. Training images captured from the screen while playing had a lot of details which did not directly help our model building process (for example: sky, trees, mountains). This additional data would require more computing power for training our dataset.

Hence, randomly selected images were subjected to the image zooming operation wherein unessential details were reduced by zooming in on the road.

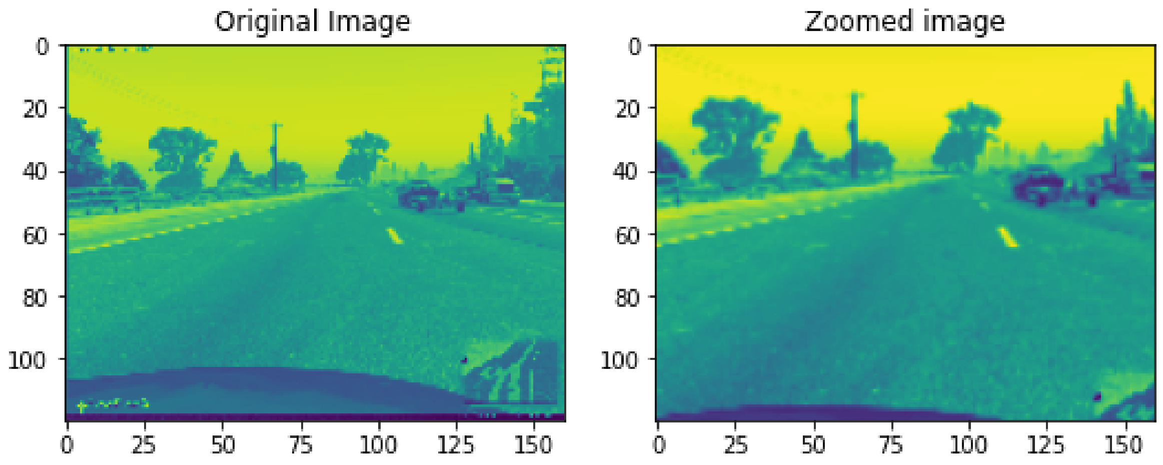


Fig 6: Image Zooming

Next, we applied the random panning operation. Panned images give a feeling of movement by blurring the background and focusing on the subject. The following figure shows the result of a panned image:

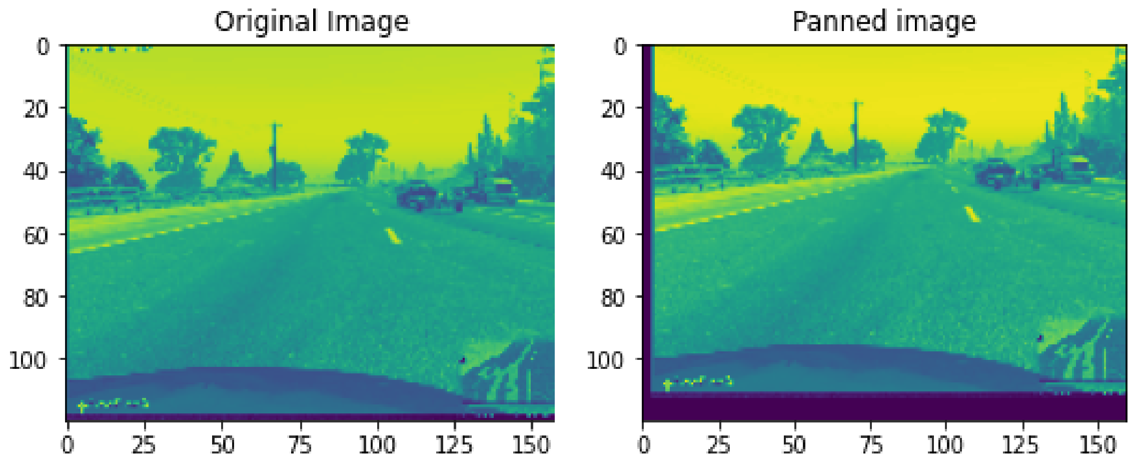


Fig 7: Image Panning

We also incorporated different brightness conditions by changing the brightness of randomly selected training images. This ensured that the training data was realistic. Images were then resized to 160x120 before feeding it into our NN architecture.

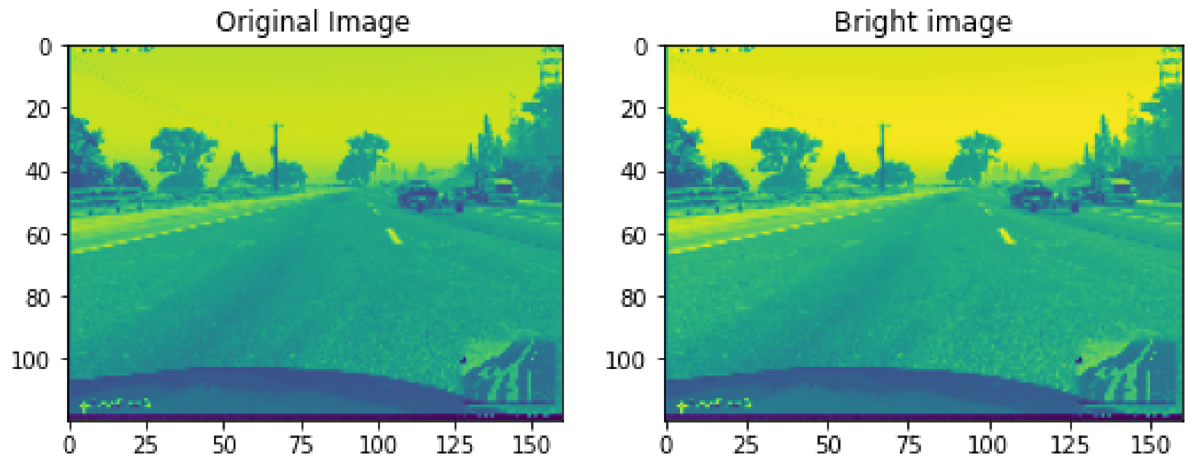


Fig 8: Image Brightening

Results

The plot of MSE Loss with respect to varying epochs helps us draw the following conclusions:

1. Epoch 1 has a train loss of 3.97, validation loss of 0.17 and as we move further along the x axis which indicates an increase in number of epochs, there is a decrease observed in both train and validation losses.
2. The final epoch 20 records a train loss of 0.1342 and a validation loss of 0.1368.

Note: Limited number of epochs have been considered due to computation restrictions.

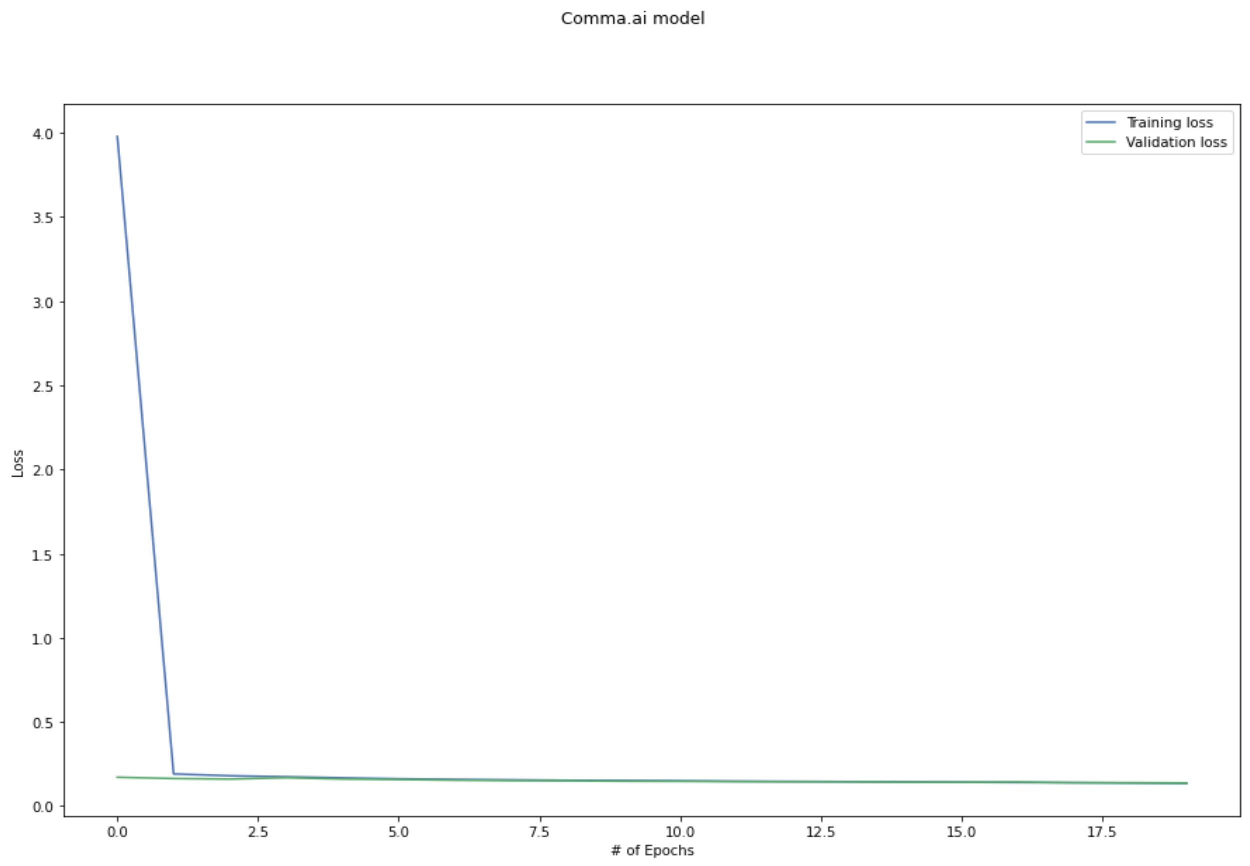


Fig 9: No. of Epoch versus Loss for Comma.ai Model

For the modified comma.ai model, we saw that the train and validation loss for epoch 1 were 0.22 and 0.16 respectively. These decreased to reach values as low as 0.08 and 0.101 for train and validation loss at epoch 20. We were clearly able to notice that there was a huge performance improvement over the baseline comma.ai model.

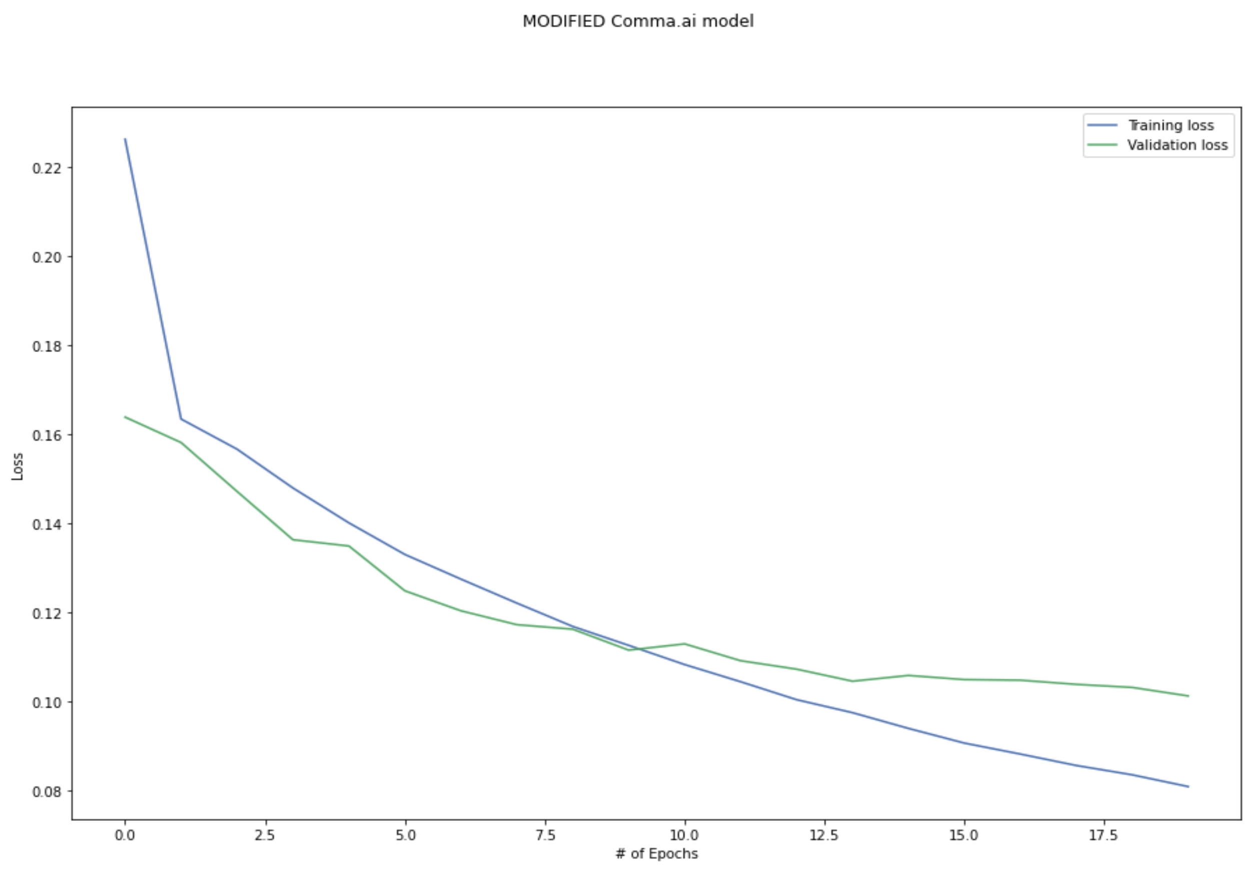


Fig 10: No. of Epoch versus Loss for Modified Comma.ai Model

The plot for our final attempt made it evident that we obtained a smooth validation loss curve which decreased over time with an increase in number of epochs. The values of train and validation loss for epoch 20 were quite similar with the second approach mentioned above. But the testing results obtained were much more accurate with the final Nvidia model. Hence, our selection was justified. It was difficult to precisely point out the reason for the modified comma.ai's unsatisfactory performance over foreign scenarios of GTA-V tracks.

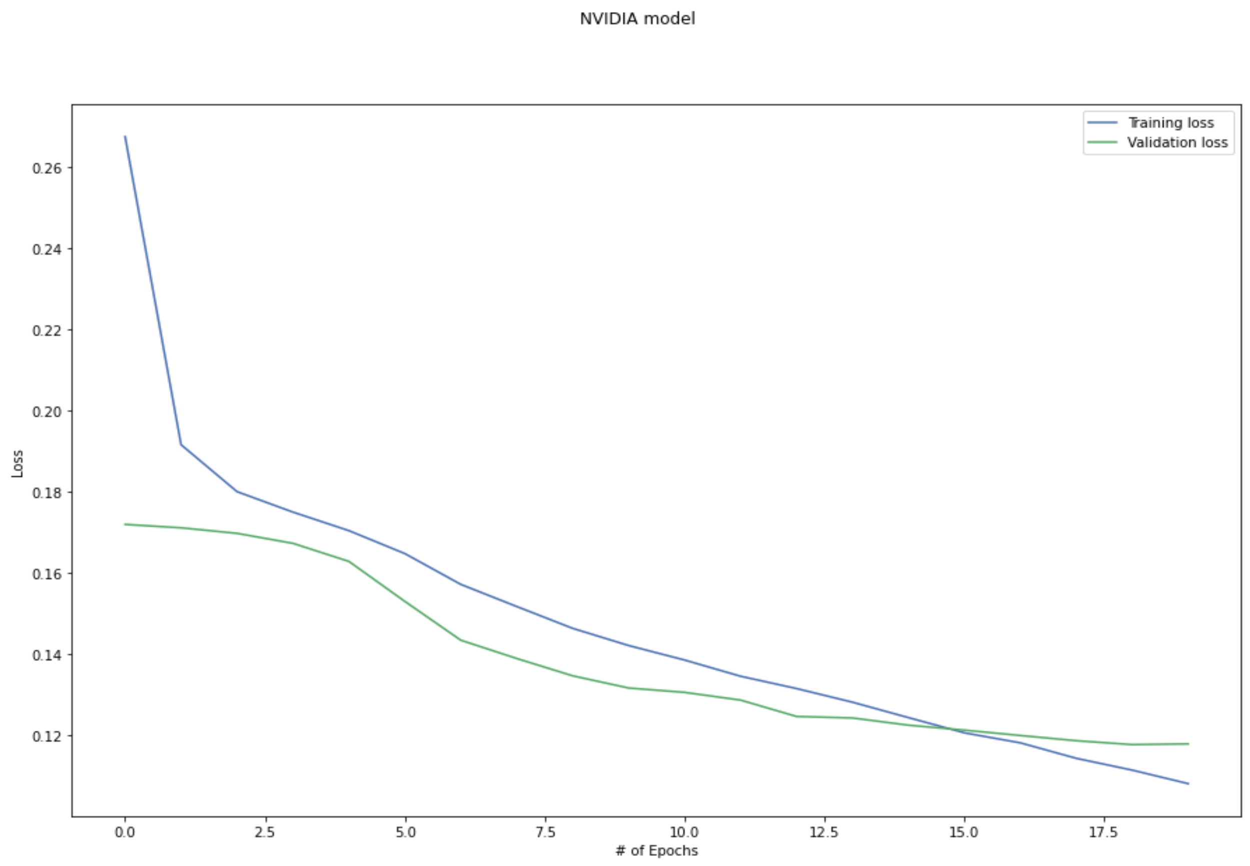


Fig 11: No. of Epoch versus Loss for Modified NVIDIA Model

References

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., . . ., Zieba, K. (2016). Retrieved from https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf

Kacmajor, T. (2018). Teaching a deep neural network to steer a car – Behavioral cloning. Retrieved from https://tomaszkacmajor.pl/index.php/2018/06/02/behavioral-cloning/

Karovalia, A. (2020). Autonomous self driving car GTA 5 [GitHub repository]. Retrieved from https://github.com/Alzaib/Autonomous-Self-Driving-Car-GTA-5

Karovalia, A. (2020). Training data angles [Data file]. Retrieved from https://drive.google.com/drive/folders/1R787vkWaMe5nsWyLpbXTG55aUv4YteTo

Bandara, U. (2017). Behavioral Cloning [GitHub repository]. Retrieved from https://github.com/upul/Behavioral-Cloning

Owoyemi, J. (2017). CarND Behavioural Cloning [GitHub repository]. Retrieved from https://github.com/toluwajosh/CarND-Behavioural-Cloning

Santana, E., Hotz, G. (2016). Learning a driving simulator. Retrieved from https://arxiv.org/pdf/1608.01230v1.pdf

Kinsley, H. [sentdex]. (2018, October 6). Python plays: Grand theft auto V [Video playlist]. Retrieved from https://www.youtube.com/playlist?list=PLQVvvaa0QuDeETZEOy4VdocT7TOjfSA8a vJoy (Version 2.x) [Software]. Available from http://vjoystick.sourceforge.net/site/