Behavioral Cloning

Hazem Ayman 18P2696
Mariam Fawzi 18P1092

Course title

CSE 485 – Deep Learning

Dr. Mahmoud Khalil

Eng. Mahmoud Selim

Note

In this zip file you will find:

- The report pdf
- Data preprocessing code – in codes/data_preprocessing.py
- Drive code – in codes/drive.py
- Model architecture code – in codes/model training.ipynb
- Trained model of track 1 – codes/track1 model/model.h5
- Trained model of track 2 – codes/track2 model/model.h5

You can find the videos for each track on the following drive: Deep Learning Project - Google Drive

This drive includes:

- Video of track 1
- Video of track 2
- Report pdf
- The zip file

Table of Contents

Table of Figures

1. OBJECTIVE

The objective of this project is to use deep neural networks and convolutional neural networks to train a model to steer a car in a game simulator autonomously. Data is collected by driving the car in the track manually and training the neural network using this data to predict the steering angle of the car during the driving phase.

First, the simulator is used to collect data on the tracks. Second, a convolutional neural network model is built using Keras to predict the steering angle of the car from the images. Finally, the model is trained and validated.

2. FILES

The project contains the following files:

- model.ipynb (notebook that includes the model and training)
- data_preprocessing.py (script used to preprocess and augment the data)
- model.h5 (the trained keras model)
- drive.py (script to drive the car autonomously)

2.1 Simulator

The simulator used in this project can be found here Simulators.zip - Google Drive [1]

2.2 Dependencies

Anaconda is used, cv2, tensorflow, python, Keras, and cuda to train on the GPU.

2.3 Driving Car Autonomously

To drive the car autonomously using the trained model, model.h5:

1) Open the simulator and enter the autonomous mode.
2) Type python drive.py model.h5 in the cmd to drive the car.

3. DATA COLLECTION

The simulator is used to collect the data by entering the training mode. We drive around the track and record the data. The data recorded are the images and a csv file. The car has 3 cameras mounted on the left, center, and right of the car. So, there are left, center, and right images stored in the IMG folder. The csv file contains the path of the image and the corresponding steering angle.

3.1 Collecting Data

We drove the first track for 5 laps and then reversed the direction to drive in the opposite direction for another 5 laps. This removes the bias towards the left and right turn. We also drove the second track for 4 laps and reversed the direction for 4 laps as well.

In both tracks, we kept the car in the middle of the lane as much as possible to capture good behavior. We also used the mouse to control the car and guide it in the track instead of using the keyboard. This is because after trials, we figured out that using the mouse shows better steering angle thus data collected is better.

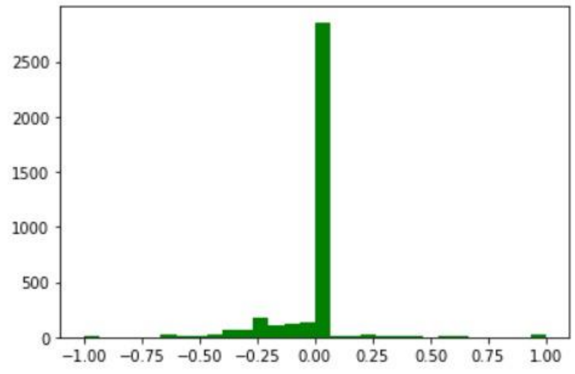


Figure 1: Data collected by keyboard

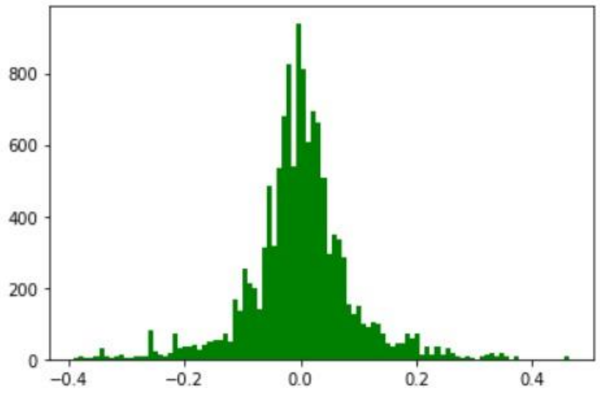


Figure 2: Data collected by mouse

3.2 Dataset Collected

We collected a total of 12.6k images from track 1 by total of 10 laps and a total of 20k images from track 2 by total of 8 laps.

4. DATA PREPROCESSING & AUGMENTATION

Data preprocessing and augmentation is a very important step in this project. It is the first step and most of the time was spent on this part and many trials were made to reach the best results. This is because the algorithm learns from the data and the output depends on how proper and clean the data is to extract the features. Generally speaking, preprocessing and augmentation are required to improve the performance, convergence, and speed up the training process. It can also minimize overfitting.

We plot a histogram of the steering angle of the data before doing any preprocessing. The histograms below show the distribution of the steering angle data for both tracks.

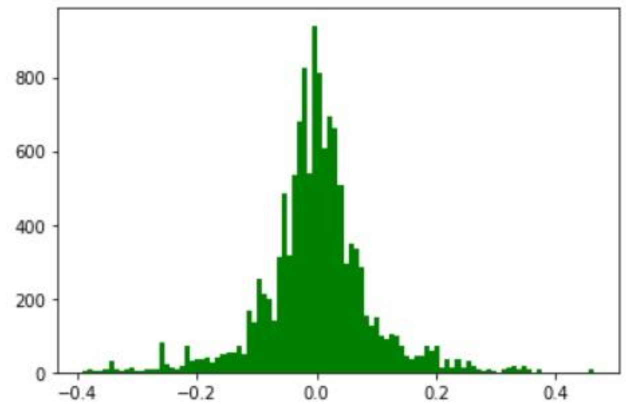


Figure 3: Track 1 data before preprocessing

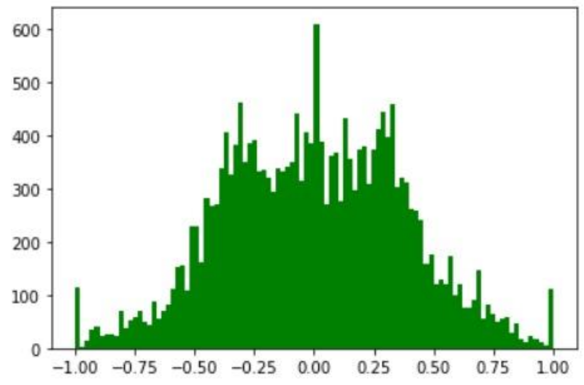


Figure 4: Track 2 data before preprocessing

4.1 Balancing the Data

Data collected is not balanced and this is shown in the histogram above of the steering angle. To balance the data, we need to reduce the number of high bins - the bins include the data of the steering angle. A histogram is plotted, and the bins are sorted to find the largest K bins. The index of the data in the largest bins are collected to be removed from the dataset. After this function, the number of data is reduced because images were removed completely. This allows having more balanced data.

4.2 Flipping the Images

The data may be biased to left or right turn. So, the trained model may show bias as well. For example, most of the turns in track 1 are left. So, the images are randomly flipped vertically, and the steering angle is multiplied by -1 based on a probability. This will also increase the diversity of the data and fools the model into having more training data.

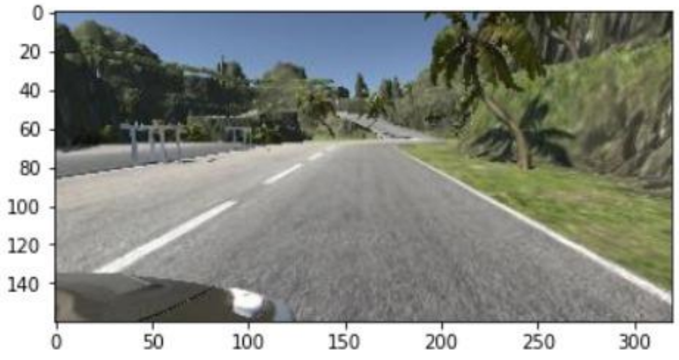


Figure 5: Original image

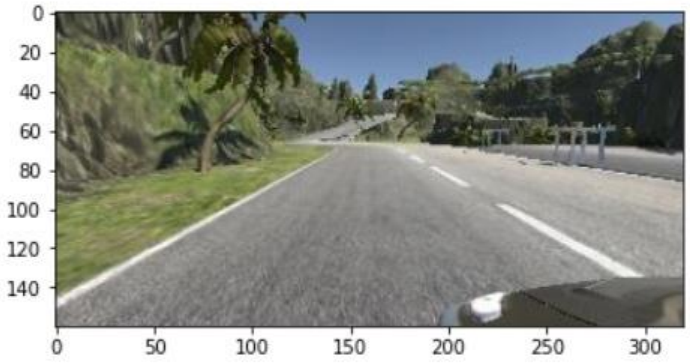


Figure 6: Vertically flipped image

4.3 Random Gamma

Gamma correction is used because different cameras do not correctly capture luminance. Gamma changes the difference between the dark and light areas. Increasing gamma will make dark areas darker and light areas lighter.

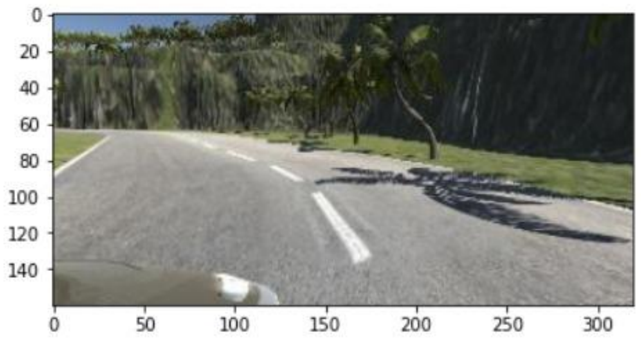


Figure 8: Original Image

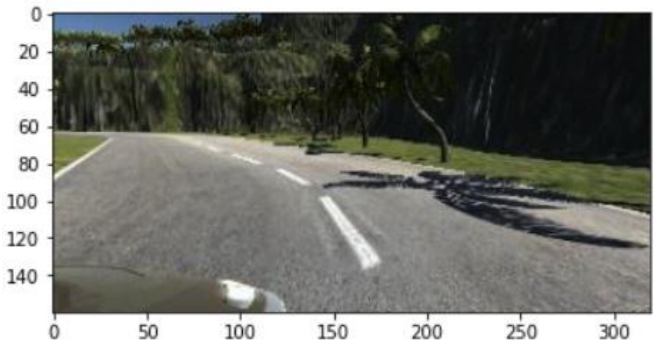


Figure 7: Gamma Correction Image

4.4 Random Brightness

Brightness is randomly adjusted in HSV space to allow learning a more general model. Also, changing the brightness is important in the second track as it has many variations in the brightness of the images so, randomly changing the brightness will allow generalizing the model and preventing it from being biased.

Brightness differs from the gamma correction function as it decreases the range of lightness without changing the starting dark point.

Brightness was used however it wasn't effective and didn't improve the results and the gamma correction factor was effective, so we didn't use brightness.

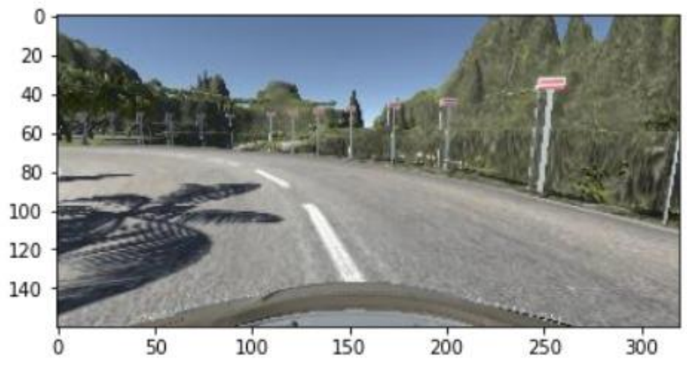


Figure 9: Original Image

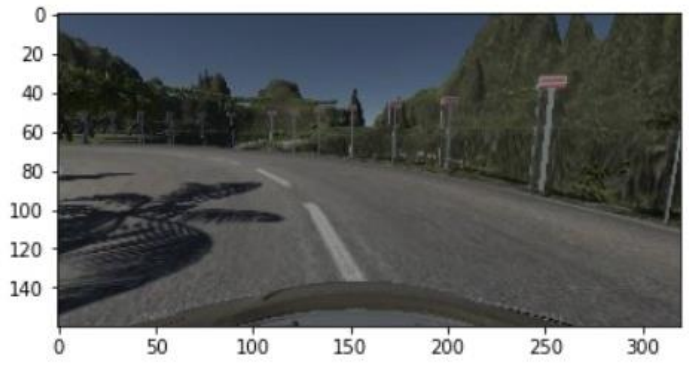


Figure 10: Brightness change image

4.5 Cropping Image

Cropping the image is important as it allows removing irrelevant information in the data. Cropping greatly affected the results. After trials, the best results were received when cropping from the top to remove the sky and from the bottom to remove the car by a specific percentage. The trials that failed included cropping the whole image from all the sides with different percentages than the current one and changing the steering angle but, this didn't give good results. We later resize the cropped image.

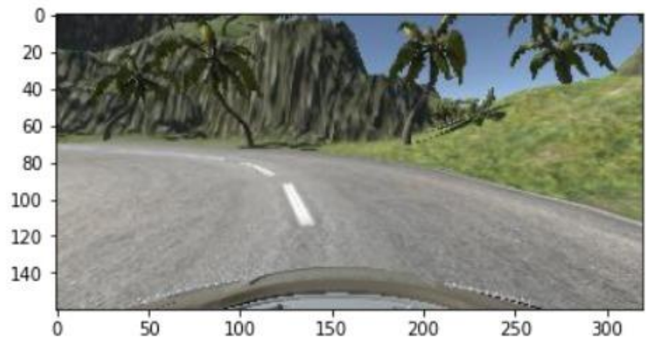


Figure 11: Original image without cropping

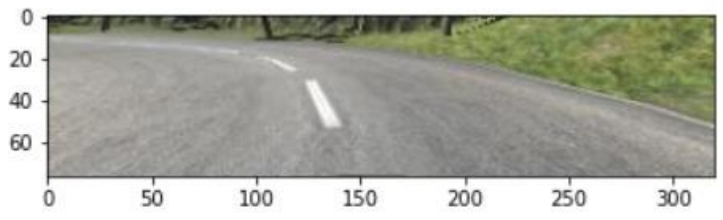


Figure 12: Cropped image

4.6 Random Shear

The images were randomly sheared horizontally. Shearing the images had the effect of making curvy roads as frequent as the straight parts in the training dataset. This was done by keeping the pixels at the bottom of the image fixed and randomly moving the top pixels left or right. The steering angle was also changed accordingly proportionally to the shearing angle. The images that were sheared were based on a Bernoulli distribution discrete random variable with a success probability of 0.8.

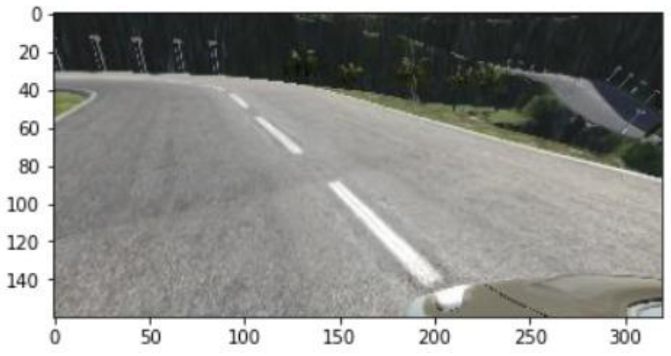


Figure 13: Original image with no shearing

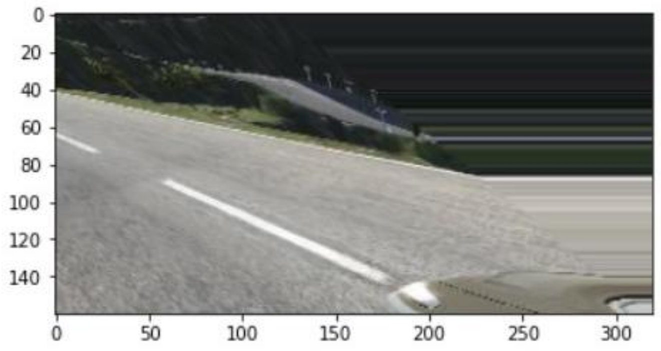


Figure 14: Sheared image (horizontally)

4.7 Subsampling the Dataset

We subsample the steering angles in the data by calculating the frequency of each steering angle and removing the angle from the dataset according to the following probability.

$$p(s_i) = 1 - \sqrt{\frac{t}{f(s_i)}}$$

<u>Where:</u>

$p(s_i)$: probability of removing the steering angle so we decided to remove ~10% in track 1 dataset and ~2% in track 2 dataset based on the probability

t: threshold which is 1e-6 in our model

$f(s_i)$: frequency of steering angle i

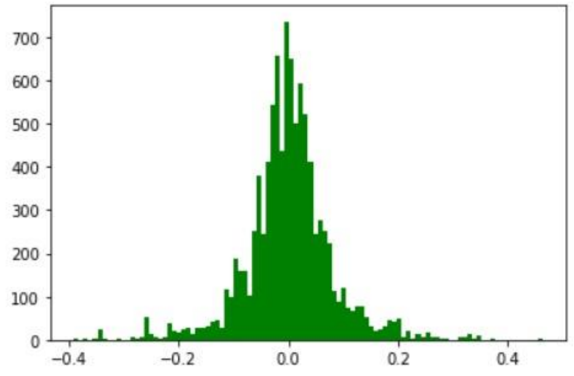


Figure 16: Data of track 1 after subsampling

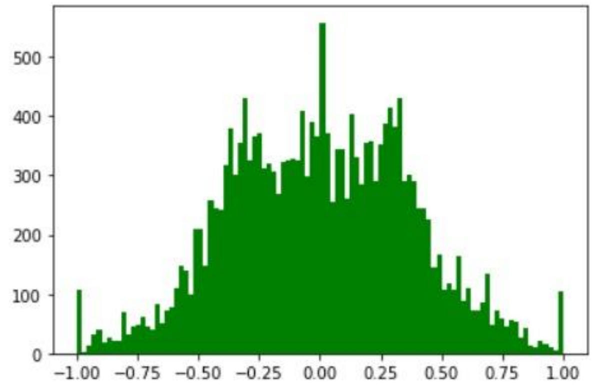


Figure 15: Data of track 2 after subsampling

4.8 Generator

This function is to load the data in memory. Since the training data collected is huge, loading it in the memory in one shot will require a huge amount of RAM. Therefore, the generator function processes only a set of images which is the batch size. This function processes the batch of images and delivers the processed data for the model to train.

Also, since there are 3 images, the steering angle captured is for the center image, therefore the left and right images steering angle are calibrated by a value of 0.222 for angle correction. If the image is left, 0.229 is added to the steering angle, if the image is right 0.222 is subtracted from the steering angle. These values are deduced from the following equation [3]:

$$\boldsymbol{s' = s + (LC)/h}$$

<u>where:</u>
s': corrected steering angle
s: the current steering angle
L: position of left camera
C: position of the right camera
h: constant recovery distance, assumed to be 4.5m
LC: distance between the left of the camera and the center of the camera which is~1.0 m

Hence, when the left camera is used **s' = s + (LC)/h=s + ¼.5** . When the right camera is used **s' = s - (LC)/h=s - ¼.5 .**

Then, a Bernoulli distributed discrete random variable with probability of success 0.8 is generated. If this variable is 1, the image is sheared randomly. Then, preprocessing of the image is done by cropping the image, flipping it, random gamma, resizing the image to be 64x64 after all the preprocessing and we finally changed the images from RGB to HSV.

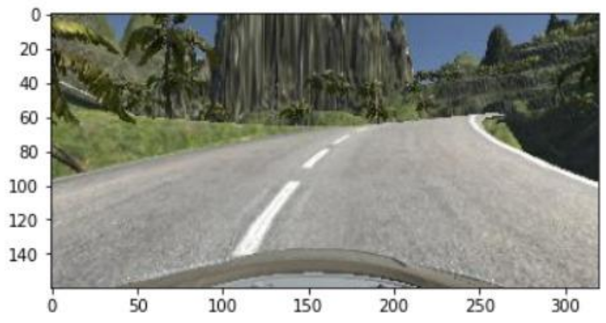


Figure 17: Original RGB Image

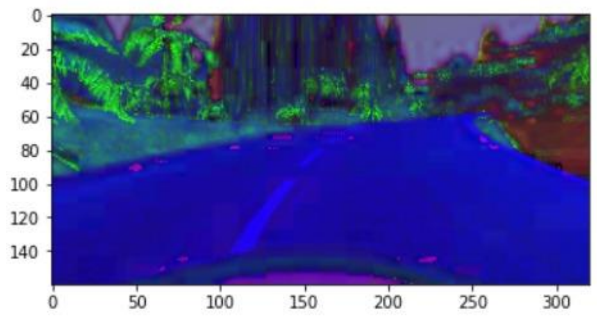


Figure 18: Converted HSV Image

Now, we have batch_x which are the processed images ready to be used by the model and batch_y which includes the angles of each image. And since it's generator function, we use yield.

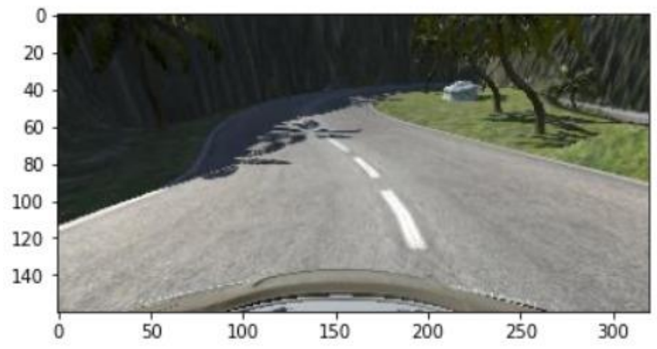


Figure 20: Image with no preprocessing

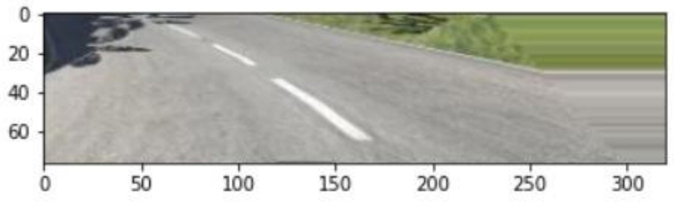


Figure 19: Image after all preprocessing

And the following histograms are the distribution of the data after preprocessing the images.

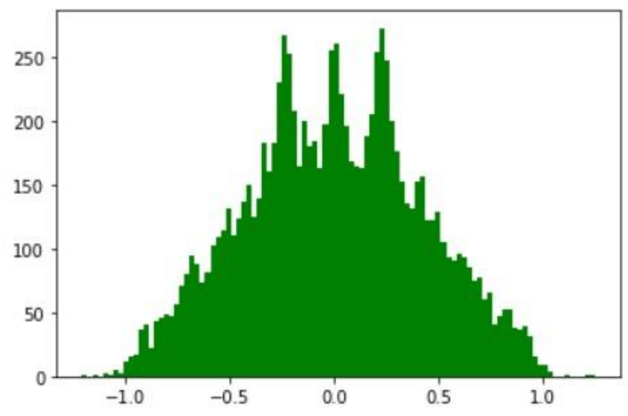


Figure 21: Track 1 distribution after preprocessing images

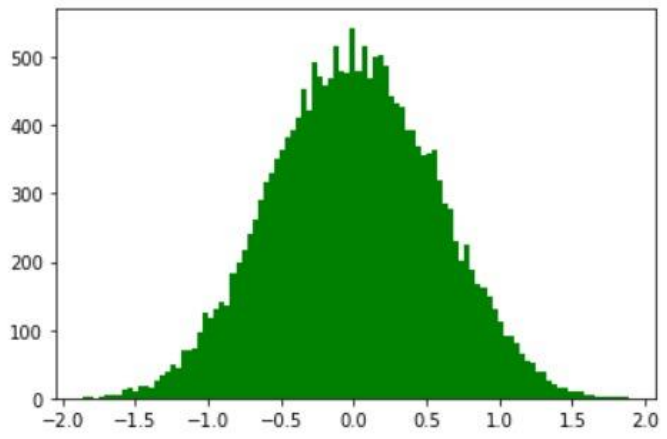


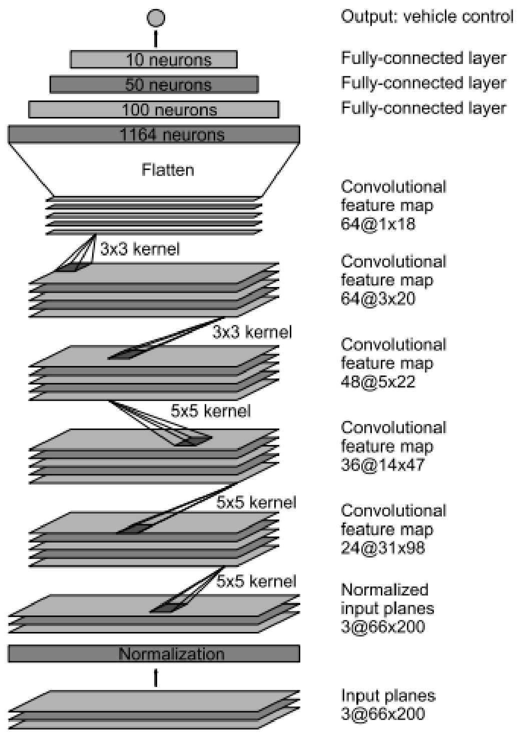
Figure 22: Track 2 distribution after preprocessing images

5. MODEL

The neural network architecture was based on the Nvidia paper "End to End Learning for Self Driving Cars". [2] The architecture consists of a normalization layer, 5 convolutional layers, and 3 fully connected layers. The model was built using tensorflow and keras.

The first layer is the image normalization in the lambda layer. Adding normalization in the model architecture itself allows accelerating the scheme by GPU processing and allows the normalization scheme to be altered with the network architecture.

Extracting the features from the images is done by the 5 convolutional layers. The first 3 convolutional layers have kernel size 5x5 and stride 2x2 with no padding (padding = valid). The last 2 convolutional layers have kernel size 3x3 and no stride (stride = 1) and no padding. After each convolutional layer there is a drop out layer with probability of 0.2. The dropout layers prevents overfitting by nullifying the contribution of some neurons.



The 5 convolutional layers are followed by flattening the data and then by 3 fully connected layers that lead to the output which is the steering angle.

Finally, the dropout layers were added at each layer in the network. It prevents overfitting by nullifying the contribution of some neurons. There is also a ReLU activation function after each layer. The purpose of an activation function is to introduce non-linearity into the output of a neuron.

First, an ELU activation function was used and when we tried a ReLU activation function, it produced better results. Therefore, we used a ReLU activation function after each layer.

Also, the dropout layer was removed, and batch normalization was put instead, however, the model didn't perform as well as the dropout layer.

The weights of the network are updated by Adam Optimizer. We used Adam as it makes the learning rate adaptive.

The number of epochs used to train the model was 30 epochs with 100 steps per epoch and training and validation batch size =64 for track 1, 256 for track 2, 512 for tuning track 2.

6. CHALLENGES

We faced many challenges and tried to overcome them with many solutions, some solutions had a clear explanation on why it's better while others were just better as per our trials. The challenges faced and how we overcame them are listed below by order:

1. When collecting the data from the simulator, we found most of the steering angle for track 1 was 0 which showed that the data was biased. This was tackled by using the mouse to control the car in the track instead of the keyboard and introducing data augmentation. This allowed having unbiased data and this can also be seen in the histograms mentioned above.
2. When augmenting the data, we thought changing the brightness of the images would generalize the data for track 2 but, when trying it didn't give good results and found that using a gamma correction function was more effective. So, we used gamma correction to overcome this problem.
3. At first, we didn't realize the importance of shearing the images but after trial, we found that shearing the images gave better results. So, we randomly sheared the images horizontally.
4. We first tried cropping the images with changing the steering angle accordingly and it gave bad results so we tried cropping the top and bottom of the image with different percentages and still didn't get good results so we finally tried cropping the top and bottom of the image with a specific percentage and got the best results and driving behavior.
5. After all this preprocessing, we still didn't get the acceptable results we wanted so, we thought of trying to balance the data and subsampling it. Subsampling allowed us to remove the frequent steering angles that don't affect the dataset and balancing the data allowed us to remove the data in the highest bins in the histogram to have a better data distribution. These together gave us great results and great driving behavior where the car stayed in the track for the whole lap.
6. Choosing the model architecture was a challenge until we found the Nvidia paper listed below and use it. Yet, we still faced the challenge of altering the model according to the problem in hand.
7. Tried adding a batch normalization layer and didn't get good results so found better results when using a dropout layer. Therefore, we used a dropout layer with probability of 0.2 to prevent overfitting and got good results.
8. We also tried adding a maxpooling and got bad results for track 2 so we didn't add a maxpooling layer in our model.
9. The Nvidia paper used ELU as an activation function after each layer but, we though ReLU would give better results. After trial, we noticed that ReLU made the driving behavior of the car better.
10. We also adding a regularizer function that is applied to the kernel weights matrix for first convolutional layer to prevent overfitting.
11. When trying the autonomous driving, the driving behavior was very bad and it showed that the preprocessing was not done correctly. We figured out we should preprocess the incoming date from the simulator, so we preprocessed the input in drive.py file.
12. Finally, track 2 was challenging as it includes many curves. However, we noticed that the steering angle is realistic, but the behavior of the car was irregular. This was because of the throttle and the PID controller mechanism was not functioning well. Therefore, we changed the throttle in drive.py from line 120.

In this code, throttle was changed based on the speed and steering angle because this indicates if the car is going uphill the speed would be decreasing so the throttle would have to increase and if the car was downhill the speed will be increasing so the throttle would decrease and the steering angle is used because if the car is in sharp turn the throttle will be decreasing according to that.

7. CONCLUSION

By the end of this project, we were able to create a model to drive the car around track 1 without going outside the lane. Another model was built to drive the car around track 2 without going outside the lane. These results can be viewed in the video provided or tried by the model.h5 file. Both models used the same architecture and preprocessing but were trained on different datasets. Model of track 1 was trained on data collected from track 1 and model of track 2 was trained on data collected from track 2 only.

8. References

[1] Simulators.zip - Google Drive

[2] https://arxiv.org/pdf/1604.07316v1.pdf

[3] Self-Driving Car Simulator — Behavioral Cloning (P3) | by Jean-Marc Beaujour | Medium