Behavioral Cloning Project 3

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed
My model consists of 5 convolution neural networks and 3 fully connected layers **(model.py lines 264-282)**.

The model includes RELU layers to introduce nonlinearity **(code line 267-271)**, and the data is normalized in the model using a Keras lambda layer **(code line 266)**.

2. Attempts to reduce overfitting in the model
The model contains 2 dropout layers in order to reduce overfitting **(model.py lines 275,277)**.

The model was trained and validated on the spilted data sets (model.py lines 191-193) and tuning with appropriate epoch and batch size parameters to ensure that the model was not overfitting **(code line 306,307).** The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track with the constant speed of 14 mph.

My note:
What have I tried out to prevent overfitting on this project?

- Get more data and re-training
- Data augmentation- flip,translate image, brightness adjustment **(model.py lines 31-89)**
- Dropout **(model.py lines 275, 277)**
- Global average pooling **(not implemented)**
- L1L2
- Early stopping **(model.py lines 326, 340)**

3. Model parameter tuning
The model used an Adam optimizer, so the learning rate was not tuned manually **(model.py line 280).** I initialised the learning rate to 0.001 on start up.
Epoch and Batch size parameter have been tuned to give the best possible result **(model.py lines 306,307)** by seeing the train and validation accuracy and result of the driving test on keeping at the middle of the road as much as possible.

4. Appropriate training data
Two set of driving behaviors have been acquired and combined for the training and validation the model.

- First is to teach the model with good driving style by keeping it on the center.
- Another is to teach the car to recover to the middle when it wheel off course to left or right

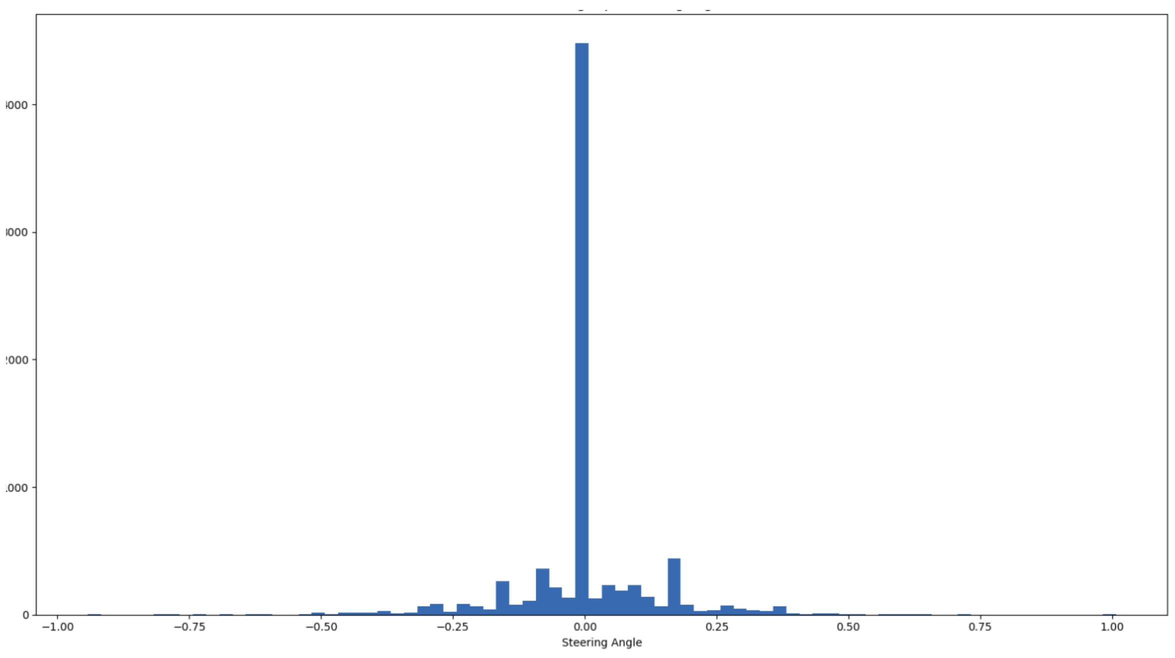
About 2 lap of good driving and another 1 lap of recovery driving are gathered separately and then combined to make a complete data set for training.

Model Architecture and Training Strategy

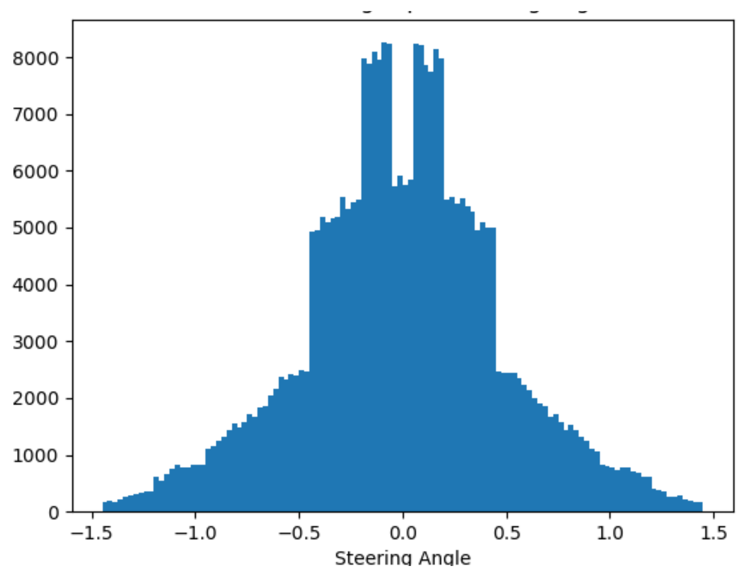
1. Solution Design Approach

My first step was to get the base model working (as suggested in the course) and use this as the benchmark for further improvement. The overall strategy for deriving a final model architecture was to start from something simple and then add more features (e.g. layers/image augmentation) to improve things.

In addition, the data was visualised to see any sign of bias. It was seen that the driving data is bias around the zero steering angles and rather negative. This will have an impact in driving straight and turninng left. So as to avoid this, I used the sampling technique to control amount of low angles data to go through the each training loop **(model.py lines 239-251)**.



Raw data



Final training data selection

My first step was to use a convolution neural network model similar to the Nvidia. I thought this model might be appropriate because it has a good number of layers but not too complex to learning the driving.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. I found that my first model had a low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was overfitting. To combat the overfitting, I modified the model with two drop out layers. Now the accuracy is on the same level.

Then I run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle nearly fell off the track. To improve the driving behavior in these cases, augmented training data with brightness adjustment, flip, adding left-right images. I also then realized that I need to help the car to learn/sense the off-track conditions and recover from it. So I add and re-train the model with this data.

Then I tune the model with the epoch and batch size parameter and rerun the simulation for the best result.

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

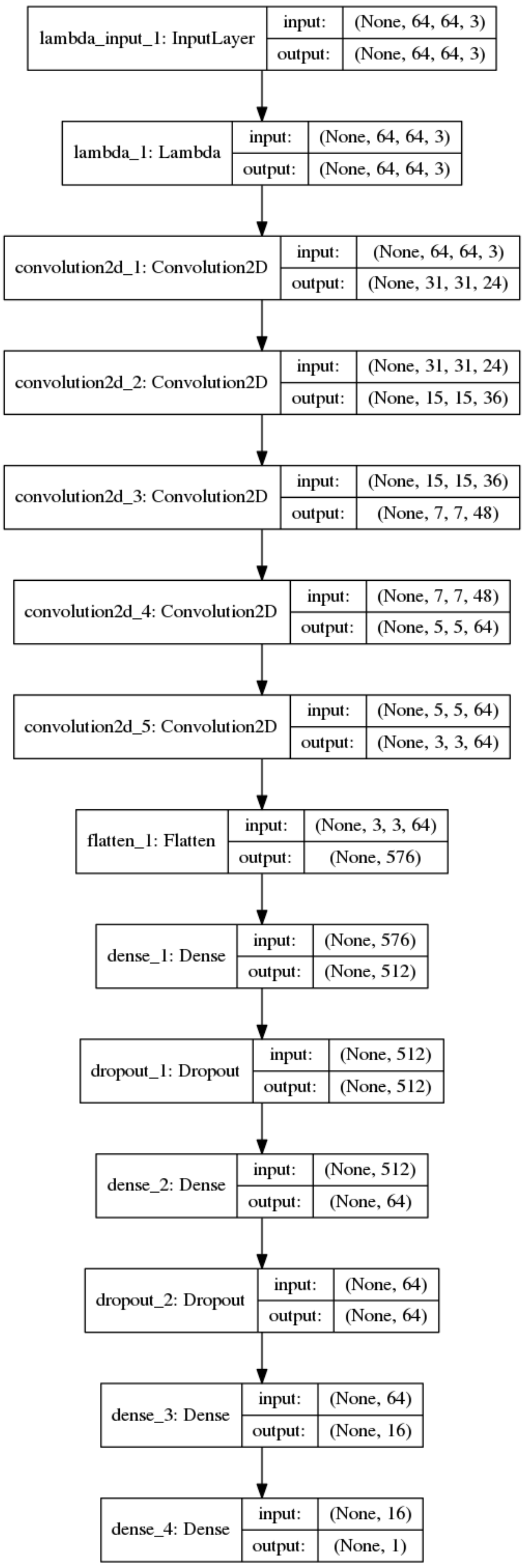
Noted that the model hyperparamters and image augmentations has been tuned for both tracks. It is a compromise. Track 1 could have performed better with slightly adjustment such as no data resize, crop more, no shadow added, no image translation, dropout less.

2. Final Model Architecture

The final model architecture **(model.py lines 264-282)** is presented below.

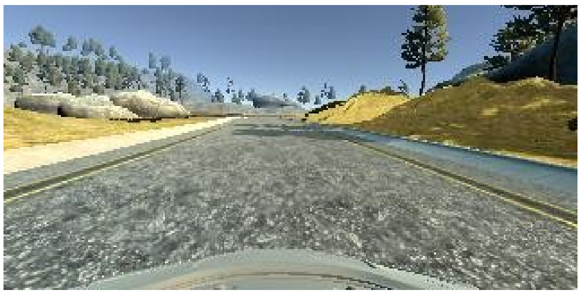
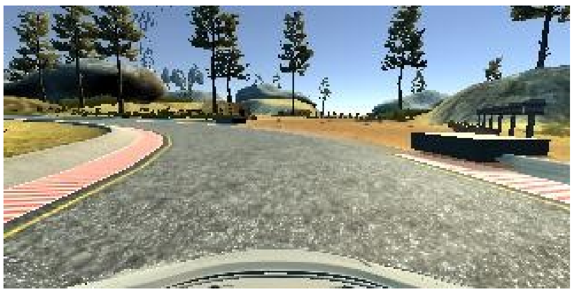
```
model.add(Convolution2D(24,3,3,subsample=(2,2),activation="relu"))
model.add(Convolution2D(36,3,3,subsample=(2,2),activation="relu"))
model.add(Convolution2D(48,3,3,subsample=(2,2),activation="relu"))
model.add(Convolution2D(64,3,3,activation="relu"))
model.add(Convolution2D(64,3,3,activation="relu"))

model.add(Flatten())
model.add(Dense(512))
model.add(Dropout(0.2))
model.add(Dense(64))
model.add(Dropout(0.5))
model.add(Dense(16))
model.add(Dense(1))
```

Here is a visualization of the architecture (note: visualizing the architecture is optional according to the project rubric)



lambda_input_1: InputLayer | input: (None, 64, 64, 3) | output: (None, 64, 64, 3)
lambda_1: Lambda | input: (None, 64, 64, 3) | output: (None, 64, 64, 3)
convolution2d_1: Convolution2D | input: (None, 64, 64, 3) | output: (None, 31, 31, 24)
convolution2d_2: Convolution2D | input: (None, 31, 31, 24) | output: (None, 15, 15, 36)
convolution2d_3: Convolution2D | input: (None, 15, 15, 36) | output: (None, 7, 7, 48)
convolution2d_4: Convolution2D | input: (None, 7, 7, 48) | output: (None, 5, 5, 64)
convolution2d_5: Convolution2D | input: (None, 5, 5, 64) | output: (None, 3, 3, 64)
flatten_1: Flatten | input: (None, 3, 3, 64) | output: (None, 576)
dense_1: Dense | input: (None, 576) | output: (None, 512)
dropout_1: Dropout | input: (None, 512) | output: (None, 512)
dense_2: Dense | input: (None, 512) | output: (None, 64)
dropout_2: Dropout | input: (None, 64) | output: (None, 64)
dense_3: Dense | input: (None, 64) | output: (None, 16)
dense_4: Dense | input: (None, 16) | output: (None, 1)

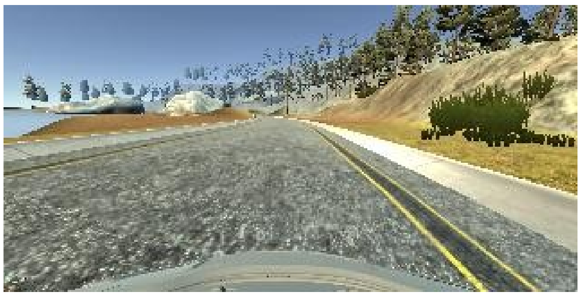
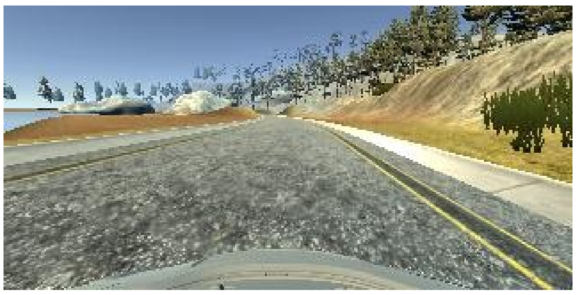
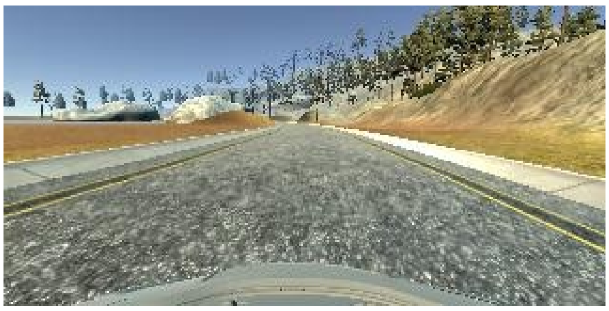
3. Creation of the Training Set & Training Process
To capture good driving behavior, I first recorded about 2 laps on track one using center lane driving. Here is an example image of center lane driving:

Straight ahead Cornering

Center mode

I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would sense and learn to drive itself back on course. These images show what a recovery looks like:

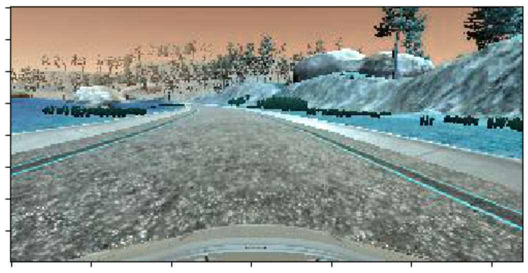




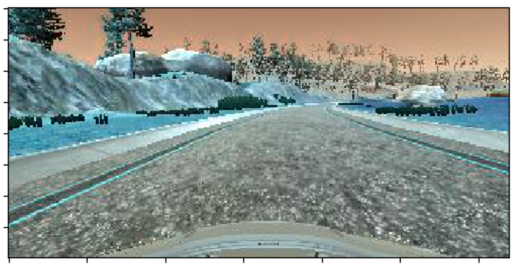
Recovery mode: Gradually turning to the center

After the collection process, I then preprocessed this data by merging two set of data together in the log file and image folder.

I finally loaded and randomly shuffled the data set and put 20% of the data into a validation set **(model.py lines 191-193)**.

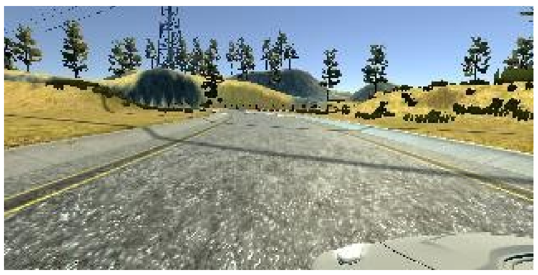
To help generalize the training data of driving only counter clockwise direction, the data is augmented with the horizontally flip images **(model.py lines 63-64).**


original


flipped

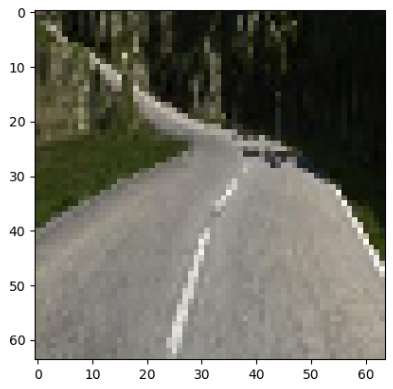
This is also helped with the use of left, right images with a fixed steering correction of 0.25 radians **(model.py lines 94-95).**

Center camera


Left camera
+ steering offset (turn ccw)

Right camera
- steering offset (turn cw)

And finally I resized the image to 64x64 pixels. I will change the scalling of the image and to avoid over fitting as well as speed up the training process.

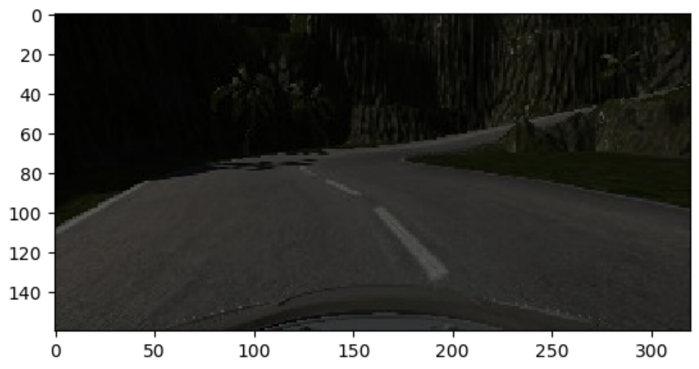


Additionally, to prevent out of memory with the big data, I use the batch training method to randomly sample and generate the batch of data **(model.py lines 220-257)**.

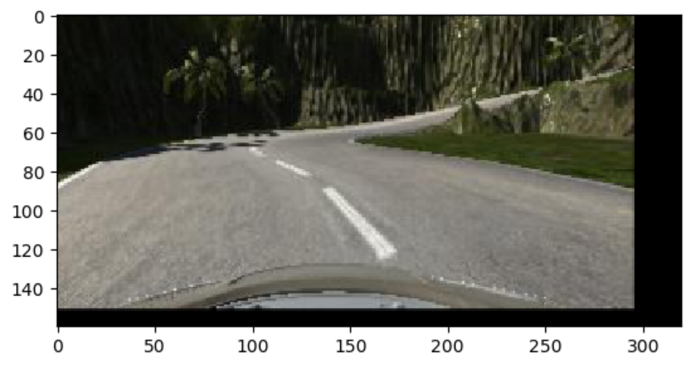
I used this training data for training the model. The validation set helps to determine if the model was over or under fitting. The ideal number of epochs was 8 as evidenced by the validation accuracy not improving further after this point. I used stop early callback to stop the training after 2 epochs not improving.

(Optional Track 2)
Then I repeated this process on track two in order to get more data points.

Track 2 is more challenging than track 1 with rather steep up and down hills, sharp bends, zig zag road and shadow overcast. To help generalise the driving in this condition without using a lot of data, I augmented the training data with image up/down, left/right translation with a corrected steering shift of 0.004 radian/pixel, and also randomly adjusting the brightness.



Brightness adjustment



Vertical and Horizontal image shift

(Racing mode)
My exploration work includes both Steering and Throttle learning as a racing mode. This is my first prototype to test out the idea. The result was not too shabby. At least the car stayed on the track1 with the variable speed (around 30 mph) I trained it to drive!
* I used the same data on track 1 to teach the car to drive with the speed as I did on the simulator.
* The work is primitive just to test out the idea. So I don't need to set the target constant speed. The car should learn itself to go fast or slow in the straight line, cornering, up hills or down hills.