Project 3: Behavioral Cloning

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

My model consists of a convolutional neural network based on Nvidia's self driving car network with a few alterations.

The model includes RELU layers to introduce nonlinearity and the data is normalized in the model using a Keras lambda layer.

2. Attempts to reduce overfitting in the model

The model was trained and validated on different data sets to ensure that the model was not overfitting. Additionally, the model was tested on the track to ensure that the car was not only driving within the plane of the track, but also very smoothly.

Data collection tricks such as driving counter clockwise with a mouse to steer, as well as using left and right image data was utilized along with recovery driving to ensure that the model was fed varied data and knew what to do to get back to the center of the track.

Also, the .csv file was analyzed to ensure that the histogram of angle frequencies (measurement variable) wasn't too biased towards the center.

The number of epochs was limited to 2 since a fairly large amount of data was used.

3. Model parameter tuning

The model used an adam optimizer hence the learning rate was not tuned manually. Other parameters such as epoch size (2) and left/right correction factor (0.2) was tuned and optimized for the best performance and minimal overfitting criteria. The model reads data from the [3], [9] and [10] columns in the .csv file, in which the [9] and [10] columns had the center measurements with a correction factor added or subtracted, depending which (left or right) camera image the code was referencing to.

4. Appropriate training data

Training data was chosen to be smooth, varied, and highly accurate. I used the mouse to steer, hence all of my turns were exceptionally smooth. I used counter-clockwise driving to prevent overfitting (similar effect if I were to choose to implement augmentation in the form of horizontal flipping). About 60% of the data was center lane driving, with the intention of preventing too much 0 angle bias. The rest of the data was recovery data, recording when the car was off center and recovering to go back to center. While utilizing left and right images should have been enough for

this task, I wanted to be sure that the car was able to recover (Also I wasn't sure I was going to be able to implement it in the time that I had).

Model Architecture Creation Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to create a robust, smooth driving experience for my virtual car. I wanted this model to not over-fit while making good use of the data that was given. I was not overfitting since my training loss and validation loss were very similar (both less than 0.007 via mean squared error)

My first step was to use a convolutional neural network similar to the Nvidia self-driving car team because of its versatility in visual data applications such as this. I figured it would be a good step to streamline the network building since this project had a lot of other factors especially pre-processing that were critical for the performance of the car. This project has given me more confidence utilizing transfer learning.

I originally had a lot of issues with my car driving off track, however once I gathered more recovery data as well as utilized left and right images with the correct correction factor, I was able to drive smoothly around the track without going off of it. The final breakthrough was to preprocess the model even further with normalization and to be conservative on the drop-out as I was doing everything else right to prevent overfitting.

2. Final Model Architecture

i.)Read in left and right images with correction factor (0.2)

LEFT **CENTRE** **RIGHT**

ii.)Append to images and measurements array and feed into X_train (independent variable feature set) and y_train (dependent variable label set), respectively

iii.)Normalize the data to ensure that it is mean centered and has the variance

iv.) Convolutional network Layers:

```
model = Sequential()
model.add(Cropping2D(cropping=((60, 25), (0, 0)), input_shape=(160, 320, 3)))
model.add(Lambda(lambda x: (x / 255) - 0.5))
model.add(Convolution2D(24,5,5,subsample=(2,2), activation="relu"))
model.add(Convolution2D(36,5,5,subsample=(2,2), activation="relu"))
model.add(Convolution2D(48,5,5,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(1162, activation=activation_type))
model.add(Dense(100, activation=activation_type))
model.add(Dense(50, activation=activation_type))
model.add(Dense(10, activation=activation_type))
model.add(Dense(1))
```

Certain sections of my code remain in my model.py remain unused as I am keeping them there in my file for future applications☺.