Behavioral Cloning Writeup Report

Behavioral Cloning Project

The goals / steps of this project are the following:

- Use the simulator to collect data of good driving behavior
- Build, a convolution neural network in Keras that predicts steering angles from images
- Train and validate the model with a training and validation set
- Test that the model successfully drives around track one without leaving the road
- Summarize the results with a written report

Files Submitted & Code Quality

1. Submission includes all required files and can be used to run the simulator in autonomous mode

My project includes the following files:

- model_nvidia_prep_lin_test.ipynb - containing the script to create and train the model
- drive.py - for driving the car in autonomous mode
- model_nvidia_prep_lin_test.h5 - containing a trained convolution neural network
- writeup_report.md - this file summarizes the results
- run1.mp4 - the video file that capture the automonomos driving on track 1 using training model above

Model Architecture and Training Strategy

1. Solution Design Approach

I first started with Instructor David's suggestion on building a simple Keras model with Flatten and Dense layers to prove a working pipeline. Once it was working, I added LeNet to be the first approval. The first result got the steering angle stucked at 25 degree and car ran off the road into the water or runnining in circle right off the start. I later figured that it was due to typo about fixated the steering angles to always positive when reading from file.

As the LeNet model trained, it was able to stay on track for a short distance before it went off road, but the car was still not capable of entering corner. I followed up with normalizing the training data set by dividing the dataset with 255 and minus 0.5. I also added a layer to crop out the noise above the road and the portion of image below the hood of the car. This helped the model to train faster as less unnecessary data to process. The improvement was not too obvious. So I updated the model to use Nvidia convulation model. However, I kept running into memory allocation issue as I memtioned earlier. As a result, I adjusted layers size with smalled output layers and using MaxPooling and dropout layers to further reduced the network size to compensate the memory allocation issue.

However, this was still not enough to improve the result significantly. I started looking into the dataset and tried to balance the dataset by looking at their distributions to trim down the excessive data points. See section Data Preprocessing for detail.

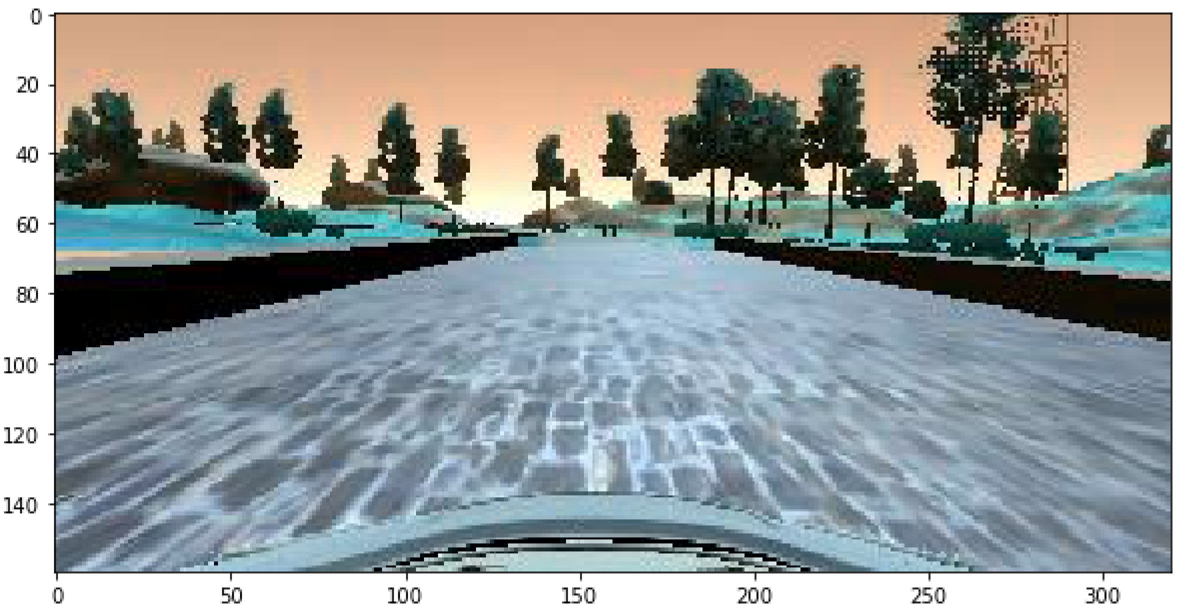
After all the improvement above, the final model was able to train the car to drive itself, and enter and exit corners. I then recorded the vehicle recovering from the left side and right sides of the road back to center. Please see run1.mpg4 for detail.

2. An appropriate model architecture has been employed

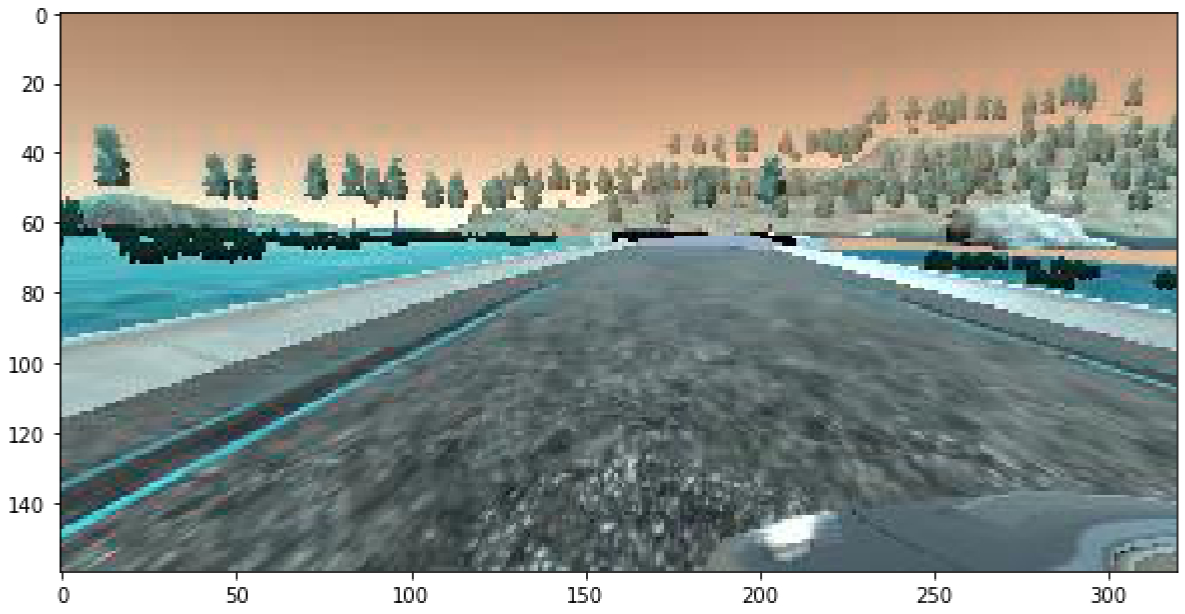
My final model consists of 3 parts: data preprocessor, data generator, and a nvidia convnet model pipeline.

date preprocessing

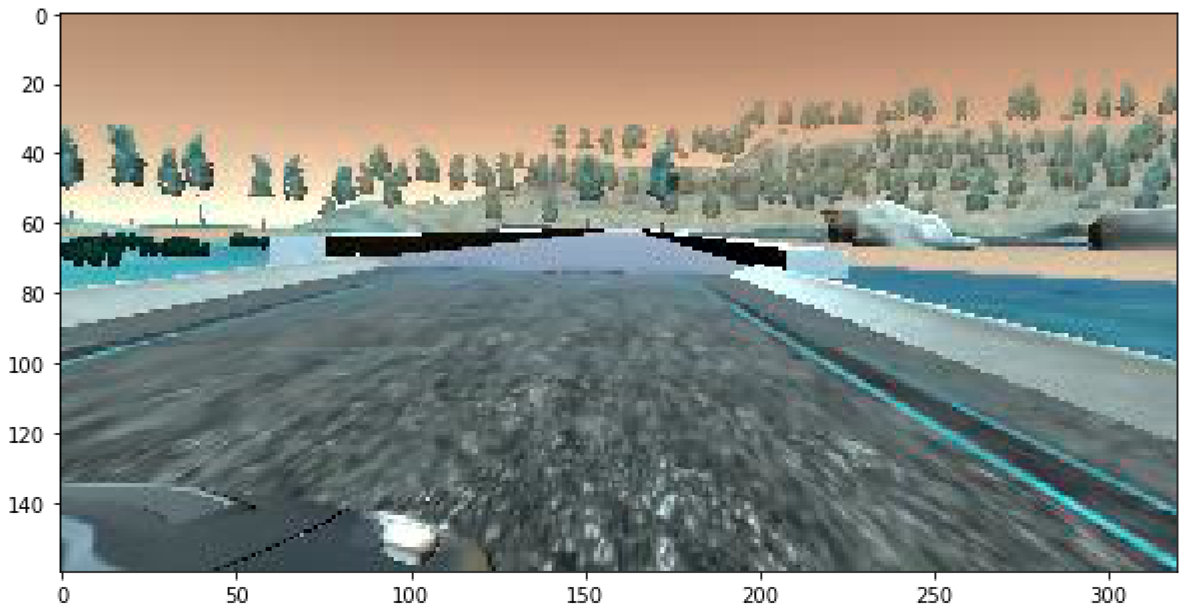
Recorded data are not necessary all useful data. Some times data of certain category may be multiple times many more then the other categories, which will lead to the trained model to become bias to the label of the majority. In this project, car appears to run striaght on open road with 0 steering angles more than making turns, which leads to heavy presents of 0 steering angle data than positive and negative steering data. The distribution below shows the imbalance of the dataset. Our goal is to preprocess the dataset, so that it has a health distribution of every data labels or steering angles to avoid overfitting.



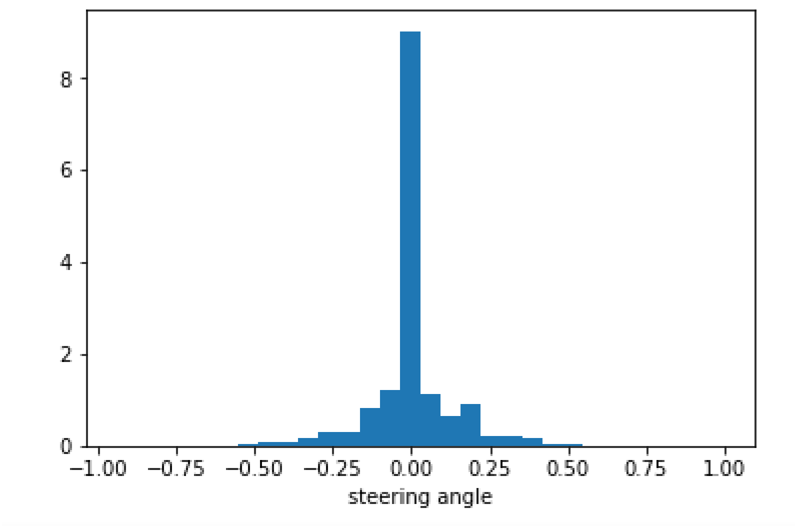
Center Image



Left Image



Right Image

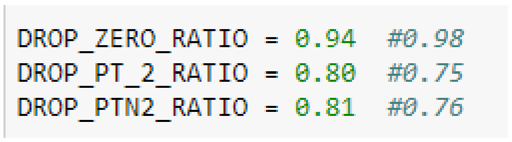


Data Visualization

From the distribution, it showed a spike at zero, which has 4000+ samples on 0 steering angle. This represents slightly more than half of the total dataset used. Between +/- 0.25 to 0 has a normal distribution. The remaining of steering angles shows a plateau at both edges on the side. To make it easier, I separated the samples into 5 bins:

- samples_00 - zero steering angles,
- samples_02 - zero-to-pos_0.5 steering angles
- samples_12 - neg_0.5_to_zero steering angels
- samples_99 - remaining steering angeles outside of the above three

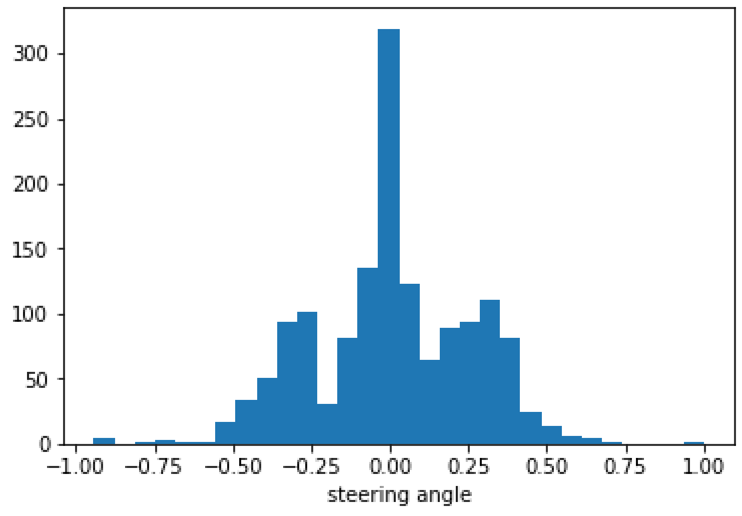
After a few rounds of trying different drop out ratio, I finalized to use the following drop out, and provided a more reasonable distribution of data lables for training. I shrinks the dataset from 8000 data labels of steering angles to about 1664 useful data labels. The dataset can be futher improve to become a balance dataset. Given that the final result is satisfactory, I stopped balancing the dataset.



```
DROP_ZERO_RATIO = 0.94  #0.98
DROP_PT_2_RATIO = 0.80  #0.75
DROP_PTN2_RATIO = 0.81  #0.76
```

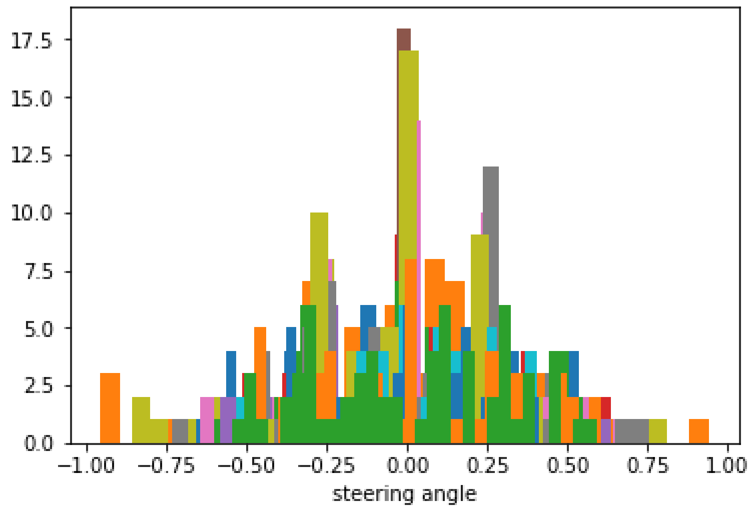
Code snippet

```
len bal_samples_00 261
len bal_samples_02 313
len bal_samples_12 279
len     samples_99 636
len samples 1489
```



Data Visualization

The submission version provides the following distribution of dataset for training.



Data Visualization for training dataset

data generator

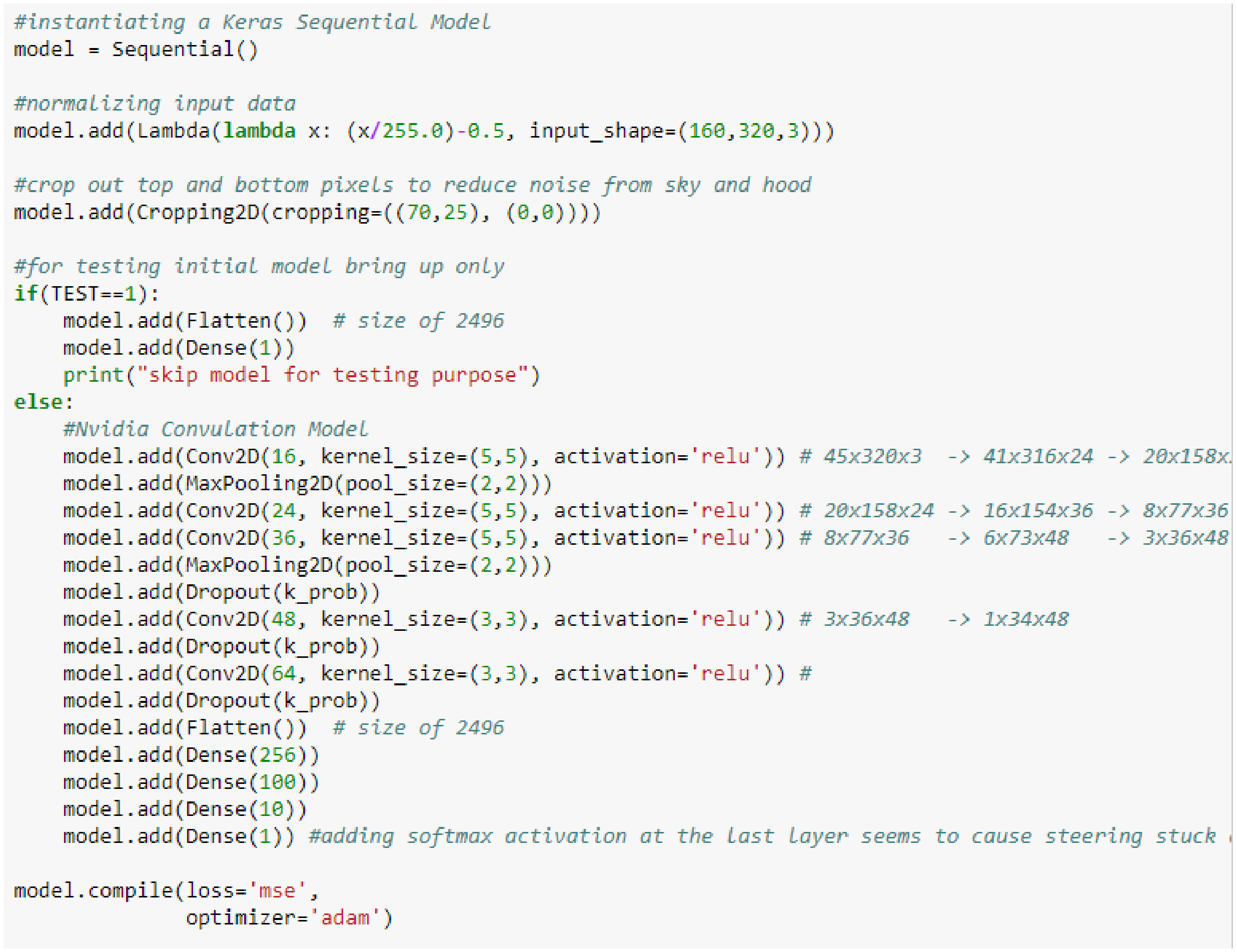
Taking the suggestion from Instructor David, I am using center, left and right images, with left and right images adding and subtracting a small correction factor to the steering angles to create another set of data for training. I also use the flipped center image to create a mirror image. This increase the improved 1664 data labels by 4 times to 6656 data labels for training and validation dataset. This is a large dataset to feed through the convoulation network. To improve the efficiency, I used generator to yield data at each batch size to avoid loading all data to memoary at once.

```
for batch_sample in batch_samples:
    #processing center image
    name = 'data/IMG/'+batch_sample[0].split('/')[-1]
    center_image = cv2.imread(name)
    images.append(center_image)

    #processing steering angle
    center_angle = float(batch_sample[3])
    angles.append(center_angle)

    #processing flipped center image
    flip_image = cv2.flip(center_image, 1)
    #flip center image and invert steering angle by multiply by -1
    images.append(flip_image)
    angles.append(center_angle*-1.0)

    # if this is for validation data samples, don't include the manipulated data below
    if(validation == False):
        #manipulate images to increase data sample for training
        #processing lef image
        name = 'data/IMG/'+batch_sample[1].split('/')[-1]
        left_image = cv2.imread(name)
        #add left image and add a random positive correction factor to steering to left
        images.append(left_image)
        #angles.append(center_angle+0.25)
        angles.append(center_angle+((1-abs(center_angle))*0.25))

        #processing right image
        name = 'data/IMG/'+batch_sample[2].split('/')[-1]
        right_image = cv2.imread(name)
        #add right image and add a random negative correction factor to steering to right
        images.append(right_image)
        #angles.append(center_angle-0.25)
        angles.append(center_angle-((1-abs(center_angle))*0.25))

X_data = np.array(images)
y_data = np.array(angles)
```

Code snippet

Nvidia Convulation Network

Nvidia convnet is my final choice of network use as it yield better result than other previous choice. This version of is not exactly the same Nvidia convnet from the paper. Due training, even I uses data generateors to yield batch size training data at a time, it still hawks a lot of memory and eventually crashed due to not enough memory allocation. As a result, I had to shrink the size of the first five convulation layers from their original size of 24, 32, 48, 64 and 64 to 16, 24, 32, 48, and 64 with inserting 2 Max Pooling layers to reduce the network size. Plus, I drop the originally 1000 outputs from the Dense layer after Flatten layer to 256 outputs instead. The result is obvious and the car was able to steer left and right to keep itself in the center of the road as well as making turns at the corners.



```
#instantiating a Keras Sequential Model
model = Sequential()

#normalizing input data
model.add(Lambda(lambda x: (x/255.0)-0.5, input_shape=(160,320,3)))

#crop out top and bottom pixels to reduce noise from sky and hood
model.add(Cropping2D(cropping=((70,25), (0,0))))

#for testing initial model bring up only
if(TEST==1):
    model.add(Flatten())  # size of 2496
    model.add(Dense(1))
    print("skip model for testing purpose")
else:
    #Nvidia Convulation Model
    model.add(Conv2D(16, kernel_size=(5,5), activation='relu')) # 45x320x3  -> 41x316x24 -> 20x158x
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(Conv2D(24, kernel_size=(5,5), activation='relu')) # 20x158x24 -> 16x154x36 -> 8x77x36
    model.add(Conv2D(36, kernel_size=(5,5), activation='relu')) # 8x77x36   -> 6x73x48   -> 3x36x48
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(Dropout(k_prob))
    model.add(Conv2D(48, kernel_size=(3,3), activation='relu')) # 3x36x48   -> 1x34x48
    model.add(Dropout(k_prob))
    model.add(Conv2D(64, kernel_size=(3,3), activation='relu')) #
    model.add(Dropout(k_prob))
    model.add(Flatten())  # size of 2496
    model.add(Dense(256))
    model.add(Dense(100))
    model.add(Dense(10))
    model.add(Dense(1)) #adding softmax activation at the last layer seems to cause steering stuck

model.compile(loss='mse',
              optimizer='adam')
```

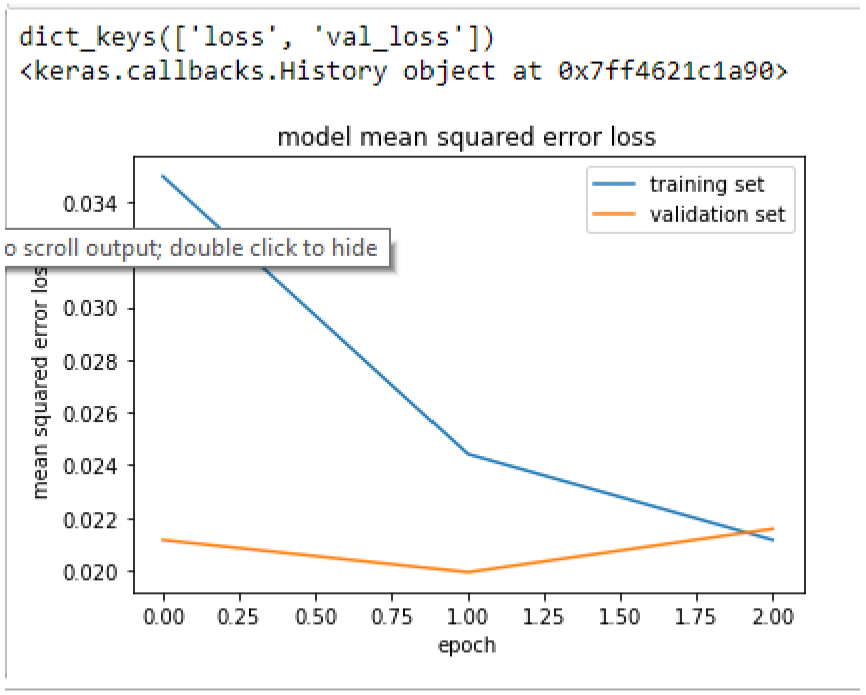
Code snippet

3. Attempts to reduce overfitting in the model

The model contains two dropout layers in order to combat overfitting.

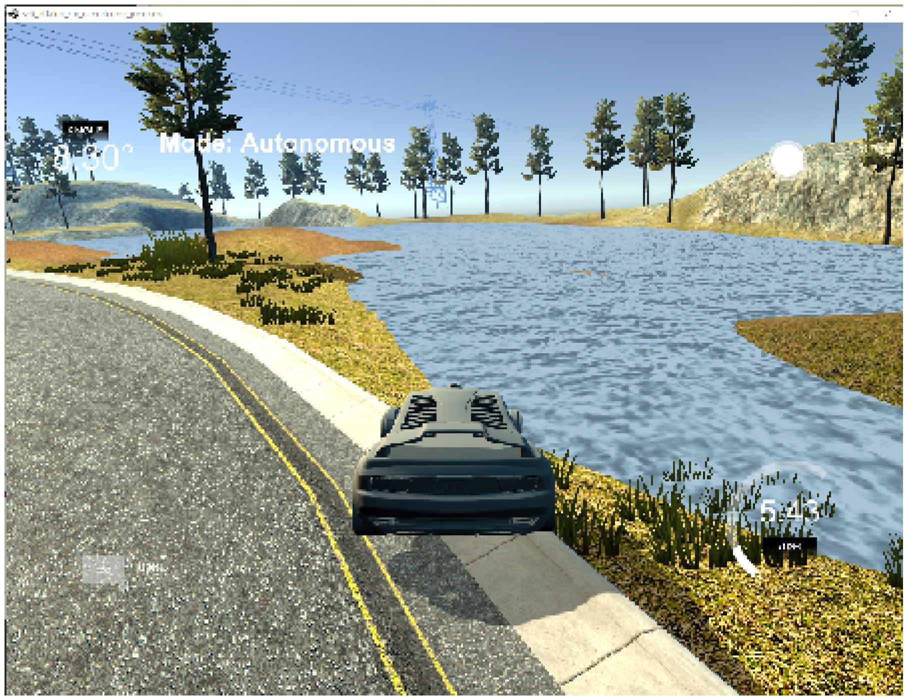
The model was trained and validated on sample data sets with 80/20 split to ensure that the model was not overfitting. The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

However, the history plot still shows validation loss bounce upward as training loss continues to improve on the 3rd epoch. During simulation, it showed the car tended to steer left and right even it was running on a striaght road. This suggested that I may have dropped to much of steering angle 0 data as the model bias on left and right steering and resulting the car crash due to steering too much.



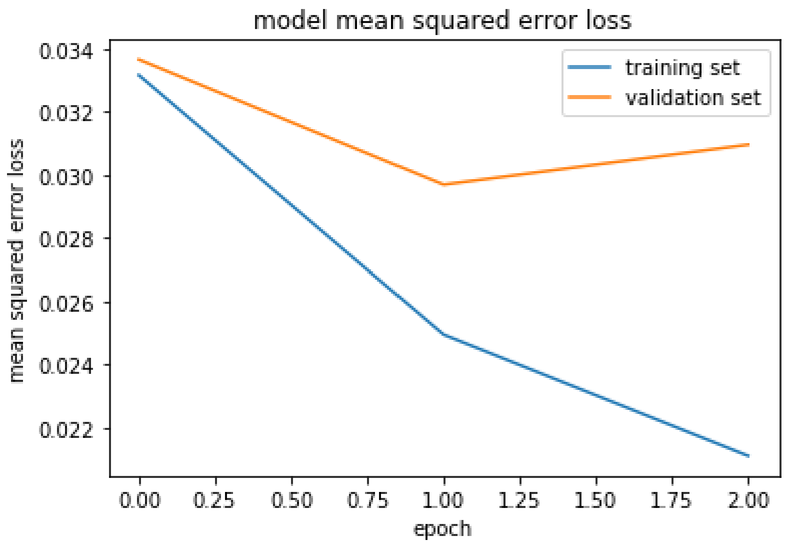
```
dict_keys(['loss', 'val_loss'])
<keras.callbacks.History object at 0x7ff4621c1a90>
```

Loss Visualization



3 epochs unsuccessful attempt

I tried adjust the parameter DROP_ZERO_RATIO to drop less zero steering angles from samples and also not to include left and right images to validation samples as their steering angles are manipulated instead of authentic. This result was worse than before the result.



Loss Visualization

Since it overfitting happened in 3 epochs, I trained the model with just 1 epoch and oberserved the best result. The trained model was able to finish a lap without going over the side lane marks. Unfortunately, the model came to a crash after 2.5 laps. More improvement can be done by finding the right balance on the dataset, or using 2 epochs, or adding more data collection for training. I will left this to future improvement.

Video

4. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually.

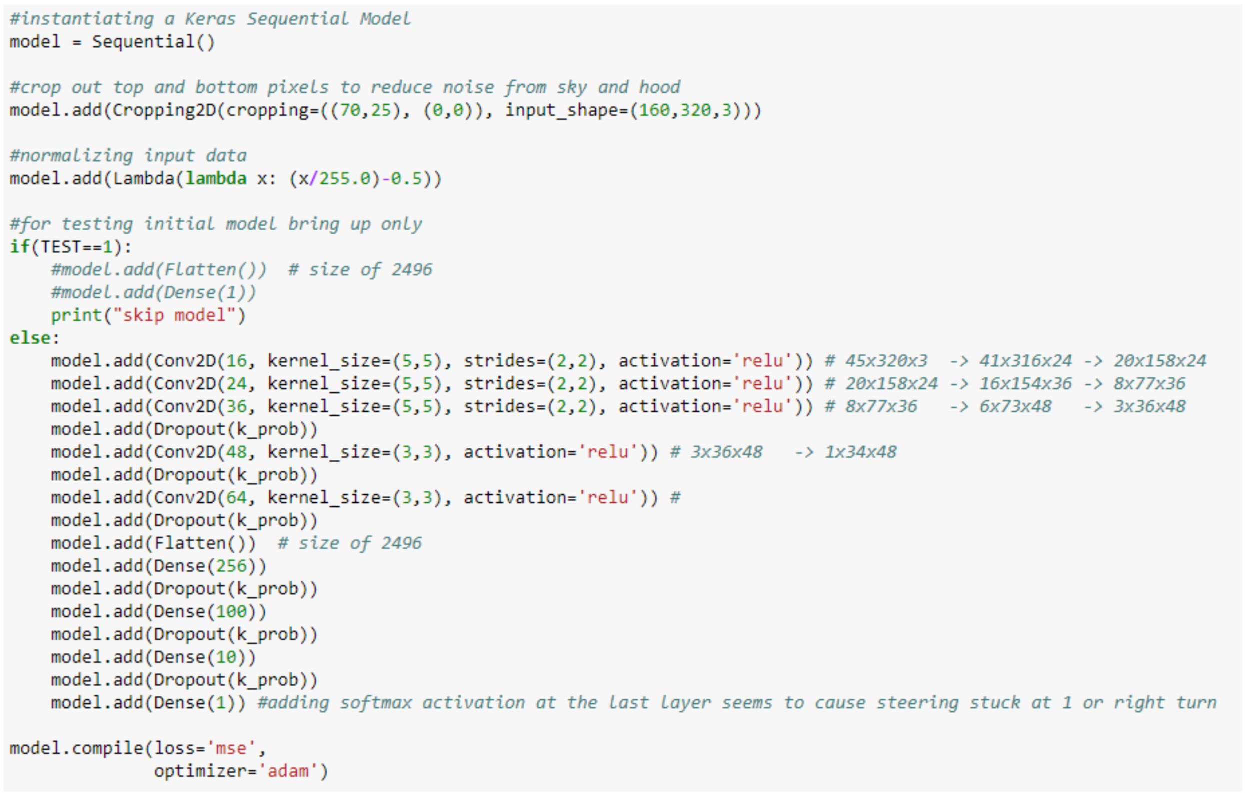
5. Appropriate training data

Training data was chosen so that the steering angles data is less imbalance than the original dataset. Please see section data_propressing for details about how I created the training data.

6. Modification:

With suggestions from reviewer after 1st submission, I cropped the images before regularization. Also it suggested that the Max Pooling layers can be eliminated by using larger strides in Convulation layer. With removing the Max Pooling layers and increasing the strides in every

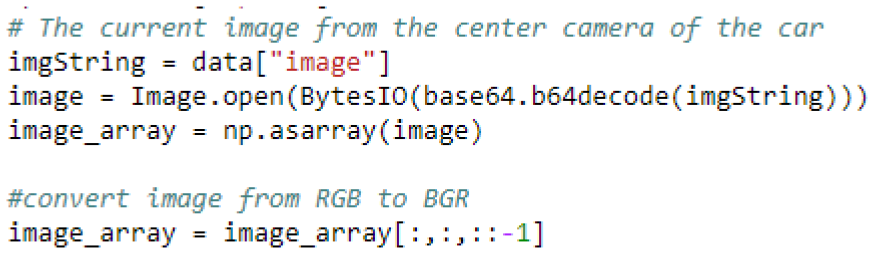
Convulation layer, it increased the efficency of the neural network and shorten the training time significantly by an hour.



```
#instantiating a Keras Sequential Model
model = Sequential()

#crop out top and bottom pixels to reduce noise from sky and hood
model.add(Cropping2D(cropping=((70,25), (0,0)), input_shape=(160,320,3)))

#normalizing input data
model.add(Lambda(lambda x: (x/255.0)-0.5))

#for testing initial model bring up only
if(TEST==1):
    #model.add(Flatten())  # size of 2496
    #model.add(Dense(1))
    print("skip model")
else:
    model.add(Conv2D(16, kernel_size=(5,5), strides=(2,2), activation='relu')) # 45x320x3  -> 41x316x24 -> 20x158x24
    model.add(Conv2D(24, kernel_size=(5,5), strides=(2,2), activation='relu')) # 20x158x24 -> 16x154x36 -> 8x77x36
    model.add(Conv2D(36, kernel_size=(5,5), strides=(2,2), activation='relu')) # 8x77x36   -> 6x73x48   -> 3x36x48
    model.add(Dropout(k_prob))
    model.add(Conv2D(48, kernel_size=(3,3), activation='relu')) # 3x36x48   -> 1x34x48
    model.add(Dropout(k_prob))
    model.add(Conv2D(64, kernel_size=(3,3), activation='relu')) #
    model.add(Dropout(k_prob))
    model.add(Flatten())  # size of 2496
    model.add(Dense(256))
    model.add(Dropout(k_prob))
    model.add(Dense(100))
    model.add(Dropout(k_prob))
    model.add(Dense(10))
    model.add(Dropout(k_prob))
    model.add(Dense(1)) #adding softmax activation at the last layer seems to cause steering stuck at 1 or right turn

model.compile(loss='mse',
              optimizer='adam')
```

Code snippet

Reviewer also pointed out a bug that the trained model used OpenCV cv2 reading images in to BGR color format while drive.py used PIL image with RGB color format. Due to the inconsistent of color format, it led to lower performance on automononus driving mode. This was fixed by converting drive.py from RGB to BGR color format.



```
# The current image from the center camera of the car
imgString = data["image"]
image = Image.open(BytesIO(base64.b64decode(imgString)))
image_array = np.asarray(image)

#convert image from RGB to BGR
image_array = image_array[:,:,::-1]
```

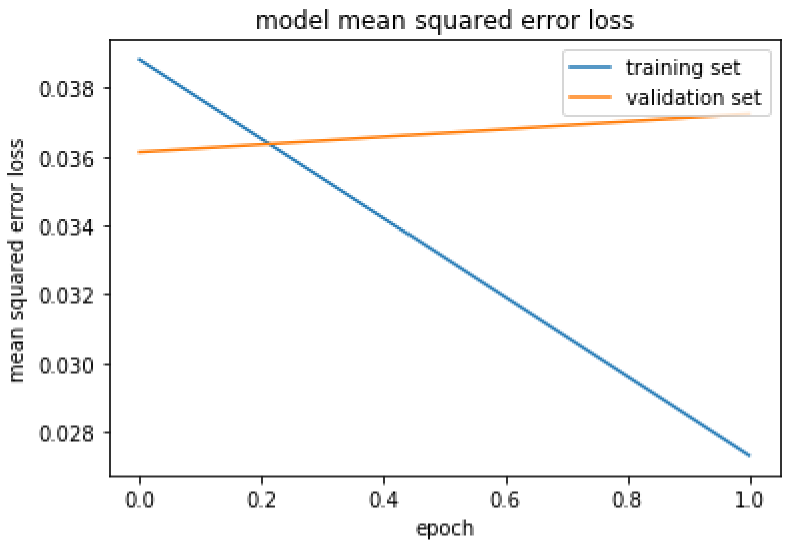
Code snippet

With just these simple fixes, my final model with 2 epochs was able to keep the car driving in the center of the road for most of the time, and made turns without stepping on the lane lines for 3 laps round. Please see attached video run2.mp4 for details.

Video

With the fixes, the model was still overfitting with the validation loss increased at 2nd epoch as shown below. Again, the overfitting could be resolved with better fine tune on preprocessing a balance dataset. I will leave that to future improvement

```
dict_keys(['loss', 'val_loss'])
<keras.callbacks.History object at 0x00000190844D06D8>
```



Loss Visualization