Behavioral Cloning

Self-Driving Car Nano Degree - Term1: Project 3

Objective:

To use the Deep Learning Knowledge learned so far to develop a **Solution using Keras**, which can learn the Driving Behavior from the images and steering angles data obtained from the simulator and steer a car around Track 1 of the simulator by meeting the criteria mentioned in the Rubric.

Approach:

1. I chose **Data Preprocessing** & **Augmentation** as the core concepts in developing the solution.
2. Most importantly, **Data Preprocessing**, since my intuition was that we can optimize the solution significantly by feeding only the relevant information like grayscale boundary needed to position the car instead of a full-fledged cropped color image which contains a lot of unwanted information being fed into the model.
3. I chose to use Nvidia's CNN Model as mentioned in the lectures as it is for the following reasons:
    a. Already proven Model so you can expect proven outputs.
    b. Any self-designed model would be fairly similar to an n-layer Convolutions followed by m-layer Flattening, so instead I preferred to spend my time and energy on analyzing the effectiveness of concepts of Data Preprocessing & Augmentation.

Learning Model:

Pre-processing: Grayscale > Gaussian Blur > Adaptive Thresholding > Crop (Top:75, Bottom:25) (60,320) > Normalization > Histogram Equalization

----------Data Generator (Batch Size:32, Epochs:5)----------

Convolutional Model Layer 1: 3 x 3 x 24 Convolution with stride of 2x2 – Activation: ReLU

Convolutional Model Layer 2: 3 x 3 x 36 Convolution with stride of 2x2 – Activation: ReLU

Convolutional Model Layer 3: 3 x 3 x 48 Convolution with stride of 2x2 – Activation: ReLU

Convolutional Model Layer 4: 3 x 3 x 64 Convolution with stride of 1x1 – Activation: ReLU

Convolutional Model Layer 5: 3 x 3 x 64 Convolution with stride of 1x1 – Activation: ReLU

Flatten: 576 Elements

Fully Connected Layer 1: 100 Elements

Fully Connected Layer 2: 50 Elements

Fully Connected Layer 3: 10 Elements

Fully Connected Layer 4: 1 Output

Loss Function: Mean Squared Error (MSE)

Optimizer: Adam

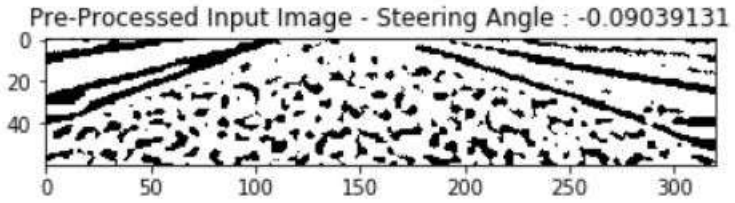
Model Summary:

```
Epoch 1/3
C:\sdcnd\envs\carnd-term1\lib\site-packages\skimage\util\dtype.py:122: UserWarning: Possible precision
m float32 to uint16
  .format(dtypeobj_in, dtypeobj_out))
C:\sdcnd\envs\carnd-term1\lib\site-packages\ipykernel_launcher.py:21: DeprecationWarning: Both axis >
- 1 are deprecated and will raise an AxisError in the future.
15060/15060 [==============================] - 468s - loss: 0.0620 - val_loss: 0.0414
Epoch 2/3
15060/15060 [==============================] - 422s - loss: 0.0375 - val_loss: 0.0355
Epoch 3/3
15060/15060 [==============================] - 482s - loss: 0.0341 - val_loss: 0.0344
```

Layer (type)	Output Shape	Param #	Connected to
convolution2d_1 (Convolution2D)	(None, 29, 159, 24)	240	convolution2d_input_1[0][0]
convolution2d_2 (Convolution2D)	(None, 14, 79, 36)	7812	convolution2d_1[0][0]
convolution2d_3 (Convolution2D)	(None, 6, 39, 48)	15600	convolution2d_2[0][0]
convolution2d_4 (Convolution2D)	(None, 4, 37, 64)	27712	convolution2d_3[0][0]
convolution2d_5 (Convolution2D)	(None, 2, 35, 64)	36928	convolution2d_4[0][0]
dropout_1 (Dropout)	(None, 2, 35, 64)	0	convolution2d_5[0][0]
flatten_1 (Flatten)	(None, 4480)	0	dropout_1[0][0]
dense_1 (Dense)	(None, 100)	448100	flatten_1[0][0]
dense_2 (Dense)	(None, 50)	5050	dense_1[0][0]
dense_3 (Dense)	(None, 10)	510	dense_2[0][0]
dense_4 (Dense)	(None, 1)	11	dense_3[0][0]

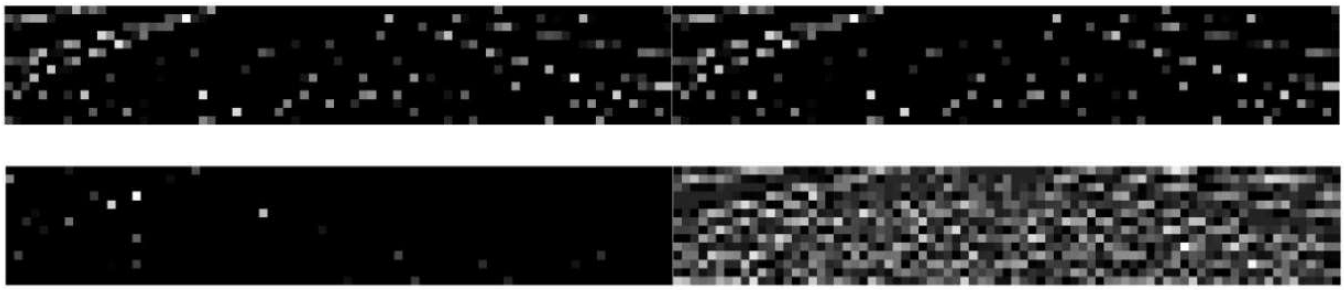
```
Total params: 541,963
Trainable params: 541,963
Non-trainable params: 0
```

Model can be Visualized as follows:



Visualizing the effect of the learned weights on an image in Convolutional Layer 1 with 4 layers of the 36 layer depth channels:

```
Layer 1 Weight: (1, 14, 79, 36)
```



Experiments:

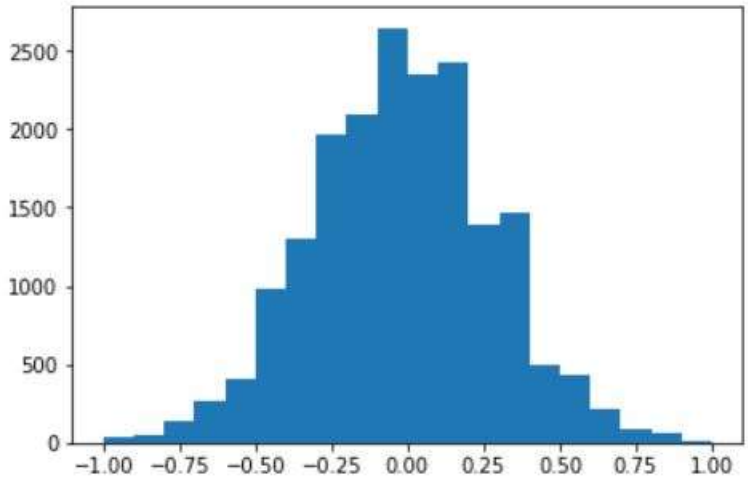
- Initial experiments were with trying to use different image pre-processing techniques like:
  - Grey-scaling
  - Gaussian Blurring
  - Contrast Stretching
  - Thresholding
  - Adaptive Thresholding
  - Bilateral Filtering
  - Normalization
  - Mean Centering

- Experimented with Batch Image Generator & Image Data Generator (augmented with custom features)
- Dropping samples with steering angle = zero

Data Analysis:

- Trained the model with these experiments and found that the model would perform well on normal straights and fairly simple turnings.
- But would enter the dirt section after the bridge and also could not navigate the tight turns after the dirt section ended.
- Also visualized the pre-processed inputs and fine-tuned it for optimum data extraction.

Fairly Distributed Steering Angles of the Augmented Input Data



Learnings:

- The Pre-Processing Layer mentioned above in the Model description was sufficient to extract the boundaries from an image.
- Overfitting [Updated for a review comment]:
    - The accuracy on the Training Set and the Validation Set were almost similar, with Validation Accuracy being slightly lower than the Training.
    - Also I had augmented the input data with a lot of additional data, to address the issues related to lack of steering angles with higher values (for tight turns and recovering from edges).
    - So I had not observed any OR run into any issues with Overfitting, and hence did not add any methods to address the same.
    - However as part of the review comment I tried to experiment with a dropout layer at the end of the Convolutional Stack (so that you lose the barely used neurons before feeding to dense layers) and its outcome was not very different to the one without it.
- The Model performed well initially until it hit the dirt section after the bridge.

- Found out that the Training Data from Udacity lacked higher steering angles which might be the cause for the above behavior.
- To overcome this added more samples of driving near the dirt section, may be a deeper model can help it avoid this with less samples.
- And after this the car would fail to navigate the tight turns 1 out of 3 times. So augmented the training set with more samples of driving the tight turns section.

To Do:

- Use the same pre-processing and try a much smaller & lighter model to understand how much optimization can be made.
- Data Augmentation: As observed from the articles on the internet, it looks like Brightness Randomization, Random Shadow Masks, Image Translations, Image Rotations can significantly improve Model Generalization. Curious to implement it.
- Try more layers in the model & Convolutional Models with more depth channels.
- Incorporate 1x1 convolutions before Dense Layers
- Use ELUs instead of ReLUs
- Try to pre-process steering angles to make the model react better to steering angles.
- Observed that the model works well with Track 1 but does not generalize as well to track 2, work on achieving it.