Project 4 – Behavioral Cloning

Write-up by Surya Vamsi for the Udacity Nanodegree Program.

In this project, the task was to train a simulated car to drive autonomously on a virtual road-like environment, using Keras functions and the concept of Convolutional Neural Networks.

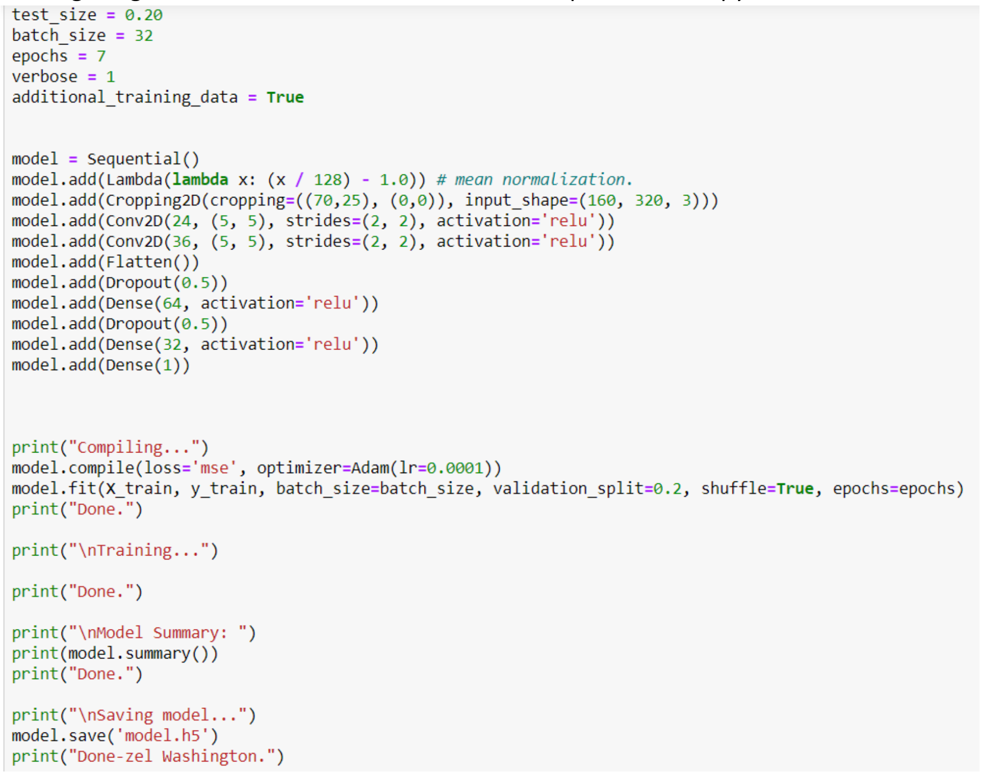
The following were the steps taken to successfully implement the project.

Step 1: Collecting images of the road by manually driving the vehicle

- The Udacity Nanodegree Simulator was utilized for performing simulation
- The Simulator was able to recognize arrow keys, so it was easy to navigate the vehicle
- Initially, the vehicle was driven on the center of the road for about 3 laps.
- Since the track was biased to the left direction, an extra lap was performed by moving the car in the reverse direction.
- Additionally, **the case of drifting sideways and then recovering back to the center of the road was also considered** and the vehicle was manually driven accordingly.
- Images captured for all cases were stored in a folder. The images were captured by a center, left and right camera, but **only the center camera was used for computational complexity purposes**.
- These images were retrieved in python on Jupyter Notebook. Data augmentation was also done so that images get flipped.
- This was done in order for generalization of the dataset, which contains images, throttle values and steering angles.

Step 2: Designing the neural network in Keras and training it.

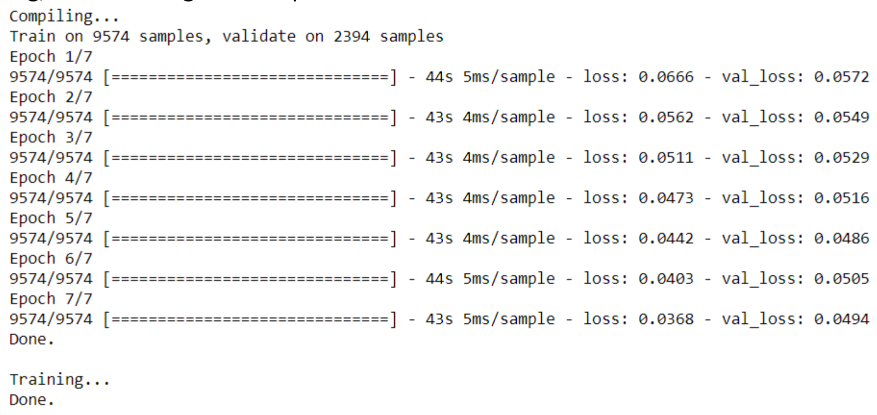
- Note: **Keras is now part of tensorflow**, hence the necessary updates were made in the main code and the drive.py file.
- The following image shows the network architecture as implemented in python



```
test_size = 0.20
batch_size = 32
epochs = 7
verbose = 1
additional_training_data = True


model = Sequential()
model.add(Lambda(lambda x: (x / 128) - 1.0)) # mean normalization.
model.add(Cropping2D(cropping=((70,25), (0,0)), input_shape=(160, 320, 3)))
model.add(Conv2D(24, (5, 5), strides=(2, 2), activation='relu'))
model.add(Conv2D(36, (5, 5), strides=(2, 2), activation='relu'))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu'))
model.add(Dense(1))



print("Compiling...")
model.compile(loss='mse', optimizer=Adam(lr=0.0001))
model.fit(X_train, y_train, batch_size=batch_size, validation_split=0.2, shuffle=True, epochs=epochs)
print("Done.")

print("\nTraining...")

print("Done.")

print("\nModel Summary: ")
print(model.summary())
print("Done.")

print("\nSaving model...")
model.save('model.h5')
print("Done-zel Washington.")
```

- As can be seen, the network initially has a preprocessing stage, where any entering image is cropped and normalized. This is done to consider only the regions containing the road curvatures, and also to reduce computational complexity.

- The network is clearly very small for this application. However, this is apparently a good performing network for this application, since the **inclusion of more layers and neurons resulted in overfitting of the data in earlier training scenarios**. (The loss would not decrease much, thus resulting in poor performance of the vehicle.)
- ***For the same reason*, model.fit was used instead of fit_generator, and also the generator function was totally not utilized in the code.**
- The test size was set to 20% of the samples, number of epochs = 7 and batch training was done with a batch size of 32. Learning rate was set to 0.0001, using the Adam optimizer.
- After training, the following was the pattern in which the loss decreased.



```
Compiling...
Train on 9574 samples, validate on 2394 samples
Epoch 1/7
9574/9574 [==============================] - 44s 5ms/sample - loss: 0.0666 - val_loss: 0.0572
Epoch 2/7
9574/9574 [==============================] - 43s 4ms/sample - loss: 0.0562 - val_loss: 0.0549
Epoch 3/7
9574/9574 [==============================] - 43s 4ms/sample - loss: 0.0511 - val_loss: 0.0529
Epoch 4/7
9574/9574 [==============================] - 43s 4ms/sample - loss: 0.0473 - val_loss: 0.0516
Epoch 5/7
9574/9574 [==============================] - 43s 4ms/sample - loss: 0.0442 - val_loss: 0.0486
Epoch 6/7
9574/9574 [==============================] - 44s 5ms/sample - loss: 0.0403 - val_loss: 0.0505
Epoch 7/7
9574/9574 [==============================] - 43s 5ms/sample - loss: 0.0368 - val_loss: 0.0494
Done.

Training...
Done.
```

-

- The model was saved to a file called 'model.h5'
- This saved model was tested on the vehicle in autonomous mode, and the result was pretty satisfactory.
- **The vehicle would however sometimes just stick to either side of the road, but it attempts to get back to the center every time, as predicted.**

Step 3: Making a video of the autonomous vehicle

- The trained vehicle has been successfully tested
- A video called video.mp4 was made and saved, which contains the motion of the vehicle around the track, for an entire lap, at 60 FPS.

INFERENCES

- *Collecting proper training data is a crucial part of training any neural network.*
- *Proper hyperparameter tuning is necessary to obtain better optimization.*

ISSUES AND DRAWBACKS

- *As can be seen from the video, the vehicle was trained and tested only on the first track of the simulator, and works well there*
- *The second track is more complicated, with inclined and declined roads, and also sharp curves. Collecting data from this scenario was a bit tedious.*

WAYS OF IMPROVEMENT

- *In order to consider the second track, a variety of scenarios such as recovery from drifting sideways on inclined right curves, left curves, declined right and left curves, deceleration when going down and acceleration when ascending need to be considered for data collection.*
- *To process all this, strong GPUs are also needed which can perform training quickly.*
- *Additional sensors such as **ultrasonic, LiDAR, Radar**, etc. will be needed for the vehicle to be more aware of its surroundings.*