Behavioral Cloning
Writeup Report – Jelena Kocić

1. Introduction

Project behavioral cloning was quite a challenge. It provide me chance to learn so much about deep learning, neural networks (LeNet, Alexnet, NVIDIA, VGG, commai approach, etc.), Keras and TensorFlow. I am sure that knowledge gained from these lessons will never be the same without project Behavioral Cloning. This amazing project force me to gather all information which I have previously learned, force me to work harder, even smarter.

2. Submitted Files

- model.py – the main file (reading data, pre-processing, NN architecture and saving the model),
- drive.py – this is original file, without changes,
- model.h5 – file where is saved my model,
- writeup report – this document,
- video.mp4 – video file with one full lap.

3. Data Collection

I run simulator model in training mode. I have chosen the lowest resolution, 320x240, and quality of video was: fastest. I decide to collect two groups of data: *mydata_small* which consists of training set of the one lap, and *mydata_big* which consists of three laps of collecting images for training set. Data was collected in directorium: /Users/Jelena/CarND-Behavioral-Cloning-P3/mydata_small/ and ../mydata_big/

I was training model and running the application from the PC, GPU 2GB. That is the reason why I had to be cautious with the amount of data. Plan was that I will make a model and train the model with the small set, in order to do that tasks faster, and to run the appropriate architecture on the big training set. That was the plan.

Example of collected images:

4. Model Architecture and Training Strategy

4.1. Reading the data

Reading data was simply achieved using functions: plt.imread, splitting the path source_path.split('/')[-1], and reading third line for the steering angle. All input images are in format (160, 320, 3).

4.2. Running basic model

During development of the model I was following the lessons. As it was suggested I started just with regular data (not pre-processed) and try with very simple network, “quick and dirty”, just to check if everything is working. That was flatten image connected to single output node to directly predicted steering angle. When I run autonomous mode in simulator which this conditions car had very interesting behavior. It was running in opposite direction, than going close to the water, but surprisingly didn’t go to the water, but car was going nest to the line between the shore and the water. That tells me that the car still learned something, how to follows the line ☺

4.3. Data Pre-processing

For data pre-processing I used:

- Normalization, using Lambda layer:

  ```
  model.add(Lambda(lambda x: (x / 255.0) - 0.5, input_shape=(160,320,3)))
  ```

- Gaining additional set of images by flipping images by vertical:

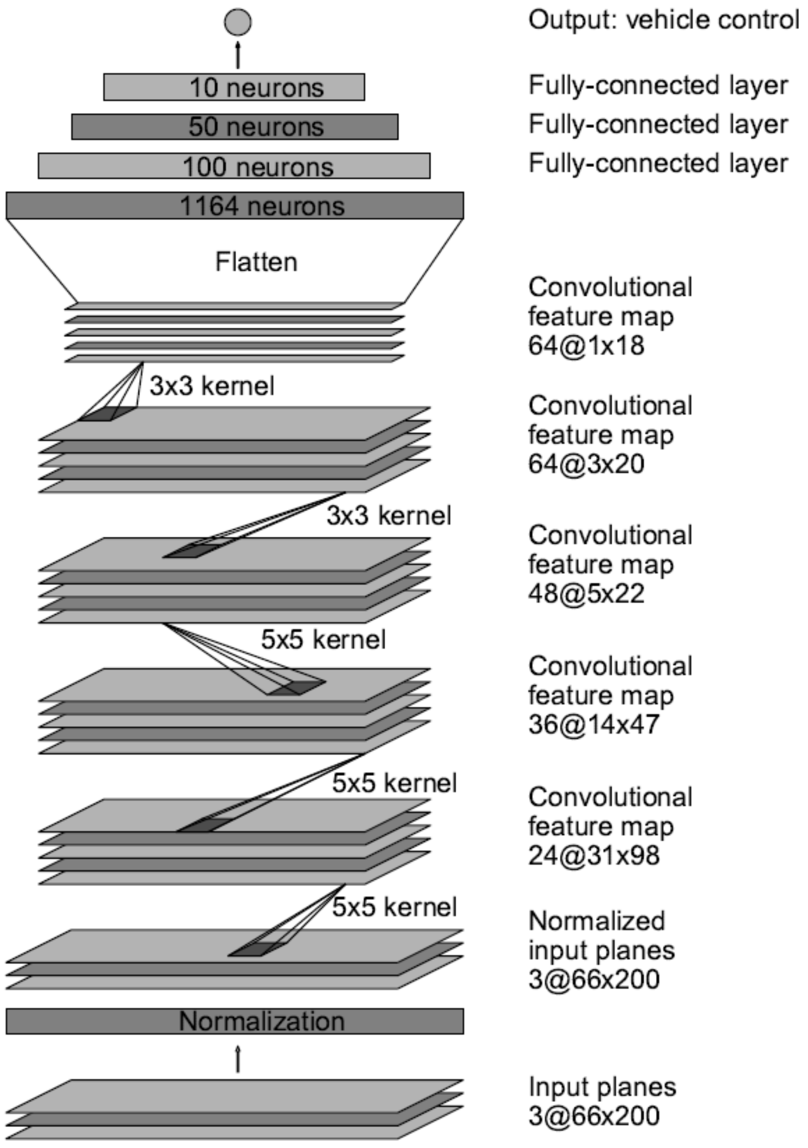
  ```
  argumented_images.append(cv2.flip(image,1))
  argumented_measurements.append(measurement*-1.0)
  ```

- Cropping picture, to remove sky, trees and part of the car on the bottom:

  ```
  model.add(Cropping2D(cropping=((70,25), (0,0))))
  ```

5. Neural Network

For the base for the neural network I chose NVIDIA’s network:



NVIDA’s network is network with 5 convolutional layers, one flatten layer and three fully-connected layers. All convolutional layers are followed with ReLU activation function. I kept original sizes of the layers:

```
# NVIDIA network model
```

```
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(100))
model.add(Dense(50))
model.add(Dense(10))
#model.add(Dropout(0.5))
model.add(Dense(1))
```

5.1. Reducing Overfitting

I have split training data in two sets, 80% for training data and 20% for validation data:

```
model.fit(X_train, y_train, validation_split=0.2, shuffle=True, epochs=4)
```

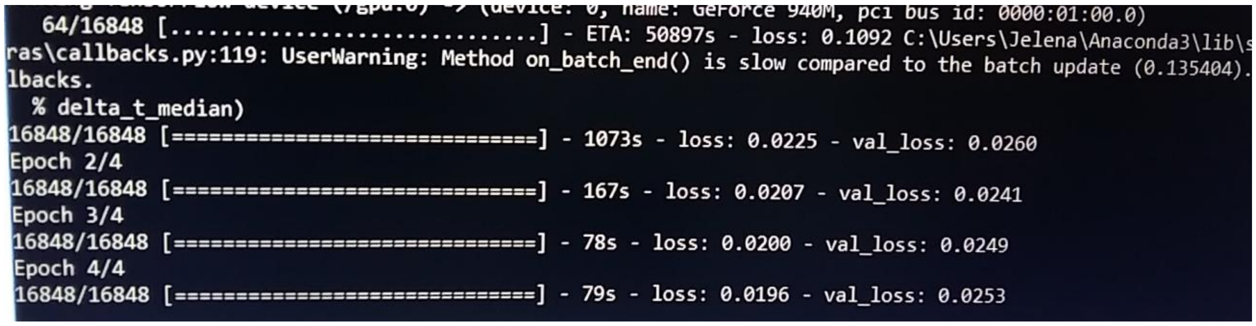
Also shuffled the data, and tried various number of epochs.

I started with 10 epochs and saw that validation loss is not increasing after 4th epoch, so that was the reason for choosing 4 epochs for the final model. Although, I have to say that I tried many different combination with number of epochs, number of input data (small and big training set) and other mechanisms for preventing overfitting, as dropout.

Example with 5 epochs:

```
6201/6201 [==============================] - 57s - loss: 0.0216 - val_loss: 0.0319

Epoch 2/5

6201/6201 [==============================] - 29s - loss: 0.0186 - val_loss: 0.0288

Epoch 3/5

6201/6201 [==============================] - 31s - loss: 0.0181 - val_loss: 0.0295

Epoch 4/5

6201/6201 [==============================] - 31s - loss: 0.0175 - val_loss: 0.0281

Epoch 5/5

6201/6201 [==============================] - 30s - loss: 0.0162 - val_loss: 0.0353
```

Example with 4 epochs:



```
                                          (device: 0, name: GeForce 940M, pci bus id: 0000:01:00.0)
   64/16848 [..............................] - ETA: 50897s - loss: 0.1092 C:\Users\Jelena\Anaconda3\lib\s
ras\callbacks.py:119: UserWarning: Method on_batch_end() is slow compared to the batch update (0.135404).
lbacks.
  % delta_t_median)
16848/16848 [==============================] - 1073s - loss: 0.0225 - val_loss: 0.0260
Epoch 2/4
16848/16848 [==============================] - 167s - loss: 0.0207 - val_loss: 0.0241
Epoch 3/4
16848/16848 [==============================] - 78s - loss: 0.0200 - val_loss: 0.0249
Epoch 4/4
16848/16848 [==============================] - 79s - loss: 0.0196 - val_loss: 0.0253
```

I have tried model.add(Dropout(0.5)) just after last fully-connected layer of 10 neurons. But the results with validation loss didn't improve, it even become slightly worse, so I eventually give up from the dropout.

5.2. Tuning the model parameters

I have choose mean square error and Adam optimizer:

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

Mean square error is chosen because this is regression network, instead of classification network as it was with traffic sigh classifier. MSE minimize difference between steering angle prediction and real steering angle.

5.3. Choosing training data

During running simulator I was trying to drive in the middle of the road all the time, in order to give good training data to my model. However, in curves, the car vas sometimes very close to the edges. In that situation I was trying to drive slowly.

For the future work, it may be good to better train car on the edges.

5.4. Data set

The idea was to make a network on small data set, and to train the final network model on big data set. There were two problems. The main problem is that big data set takes to much time to train the model! At the end when it was finished, the validation loss was not significantly better, and the car even at one point went off the road - that was second problem.

From that reason, I chose to stay with small data set even for the final model.

6. Conclusion

I succeed to autonomously drive the car for the whole track! Several times ☺ In video video.mp4 there is just one lap, but I did that several times. I succeed to apply the knowledge gained from lessons and to successfully train the network and drive the car. I am very satisfied and happy because of that.

For the future work, improvements of this model will be: using pretrain network, and train only the last layer, using frozen weights from the bottleneck. Using pictures from all cameras. Here I used only central camera. And some additional preprocessing the data can always help.