markdown

April 23, 2019

1 Project Report

My project includes the following files:

```
model.py containing the script to create and train the model
drive.py for driving the car in autonomous mode
model.h5 containing a trained convolution neural network
writeup_report.pdf summarizing the results
video.mp4
```

Using the Udacity simulator, the model thus was trained using the drive.py. The autonomous driving was recorded in the image folder run1. These images were later on compiled into a video using the video.py

Model.py shows the entire pipeline code; from reading the data, preprocessing the data and model architecture, along with the comments for the each section of the code.

1.1 Model Architecture and Training Strategy

1. **An appropriate model architecture has been employed**

My model consists of the following architecture.

```
model.add(Cropping2D(cropping=((50,20), (0,0)), input_shape=(160,320,3)))
model.add(Lambda(lambda x: x/255 - 0.5,input_shape=(row, col, ch),output_shape=(row, col, ch)))
model.add(Conv2D(24, 5, 5, activation='elu', subsample=(2, 2)))
model.add(Conv2D(36, 5, 5, activation='elu', subsample=(2, 2)))
model.add(Conv2D(48, 5, 5, activation='elu', subsample=(2, 2)))
model.add(Conv2D(64, 3, 3, activation='elu'))
model.add(Conv2D(64, 3, 3, activation='elu'))
model.add(Dropout(0.75))
model.add(Flatten())
model.add(Dense(100, activation='elu'))
model.add(Dense(50, activation='elu'))
model.add(Dense(10, activation='elu'))
model.add(Dense(1))
```

The architecture uses three layers of 5 by 5 filters with activation as ELU. Subsequently followed by Convolution layers of 3 by 3 filters. A dropout of keep probability of 0.75 is kept.

This forms the pre processing layers of the CNN. After this the inputs is flattened which is then connected to fully connected Dense layers as part the fully connected Neural Network. Elu is the chosen activation unit which is akin to ReLu, in that below zero the output from elu is non zero but a small value. And it is different from leaky Relu in that at zero, it is diffrentiable.

2. **Attempts to reduce overfitting in the model**

A dropout which achieves similar results to that of regularization is incorporated in the architecture. Which randomly kills 25% of the weights during the training epochs inorder to make the model more robust. Also model architecture was experimented with the depth of Convolutional results such that ther in sample and out of sample error are close. In the case of overfitting, the insample(training) error goes low where as the validation error goes high suggesting poor generalization. Model parameters was nearly optimized to give good generalisation.

3. **Model paramter tuning**

Since the optimzer used was adam, the learning rate hyper parameter was not tuned as adam combines both rms prop and SGD thus convergence issues are slightly mitigated. The *beta* values chosen as default values. And moreover since normalisation was carried out as a pre processing step, helps in convergence faster.

4. **Training Data**

Appropriate training data was collected for a detailed selection techniques for the data please refer to following section.

1.2 Model Architecture and Training Strategy

1. **Solution Design Approach**

I started off first with driving the car in the simulator for a lap. With the data collected, I built a simple Feed Forward Neural Network which simply flattens the image into a single vector. With that it was seen that the car was in the circuit for the first 2 seconds but then steered off in a tangent off the circuit. So to capture better data, I revistied the simulator again, carefully drove around the circuit one, 3 laps hovering around the center. With occassional induced steering towards the ledges and coming back to the center. So with good training data. I went ahead in building better architecture. Initially I started buliding convolutional layers with small depths in the order of 5,6,7 and so on. And the final architecture with online references came up with the aforementioned architecture. In the course of building the model, many times, it was witnessed that there was a significant difference between training and validation errors. To counter this, dropout was used in the model, a keep_probability of 0.75 did the trick to overcome a signifcant portion of overfitting. Collecting more data also helped in mitigating overfitting.

Along with the center images, the left and the right images was also considered. Since the left and right cameras are at a distance from the center, a distortion of 0.2 was considered as suggested. At the end of whole training process for 30 epochs, the car using the trained model *model.h5* was able to drive circuit one autonomously as reflected in *video.mp4*.

2. **Final Model Architecture**

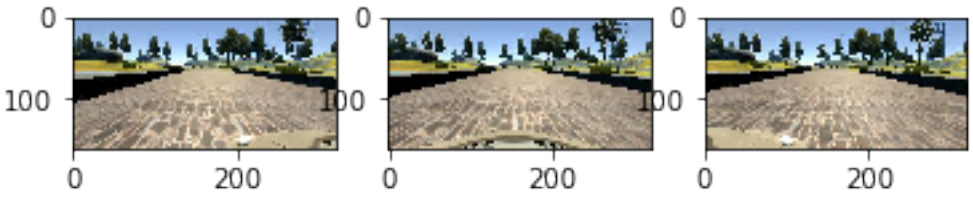
The final model is previously mentioned with its architecture.

3. ** Creation of the Training Set & Training Process **

To capture good training data, I covered roughly two and half laps around circuit one. Most of the times hovering around at the center. Also drove half a lap in the opposite direction. Some images also include steering to the ledges and then coming back to the center.
The following are a sample of images that were captured

```
In [28]: import matplotlib.pyplot as plt
         import matplotlib.image as mpimg
         leftimage = mpimg.imread("left.jpg")
         centerimage = mpimg.imread("center.jpg")
         rightimage = mpimg.imread("right.jpg")
         plt.subplot(131)
         plt.imshow(leftimage)
         plt.subplot(132)
         plt.imshow(centerimage)
         plt.subplot(133)
         plt.imshow(rightimage)
```

```
Out[28]: <matplotlib.image.AxesImage at 0x133f68a58>
```



With this we can see that recording images with multiple cameras especially at the center of the region is helpful in driving the car autonomously within the track