Use Deep Learning to Clone Driving Behavior

Required Files

Are all required files submitted?

The submission includes a model.py file, drive.py, model.h5 a writeup report and video.mp4.

1st Resubmission

My project includes the following files:
model.ipynb -> this file contains the code for creating the Keras model and training the model.
Drive.py -> this file is using for driving the car in the autonomous mode and I have change the body of this file a bit because my logic for creating the model.
Model.h5->this is the trained model

Quality of Code

Is the code functional?

The model provided can be used to successfully operate the simulation.

1st Resubmission

For driving the car in the autonomous mode of the simulator, two files are necessary:

1- Model.h5
2- Drive.py

To start driving the car the following command should be called

```
python drive.py model.h5
```

Is the code usable and readable?

The code in model.py uses a Python generator, if needed, to generate data for training rather than storing the training data in memory. The model.py code is clearly organized, and comments are included where needed.

I have commented the code the same as write-up.

1st Resubmission

The Keras's Model in the model.ipynb file is saved after training as model.h5 file, therefore it's usable. The following code snippet is used for saving the model.

```
model.save('model.h5')
```

Model Architecture and Training Strategy

Has an appropriate model architecture been employed for the task?

The neural network uses convolution layers with appropriate filter sizes. Layers exist to introduce nonlinearity into the model. The data is normalized in the model.

The layers of model are explained in one of the next sections in detail. The model consists of the following layers.

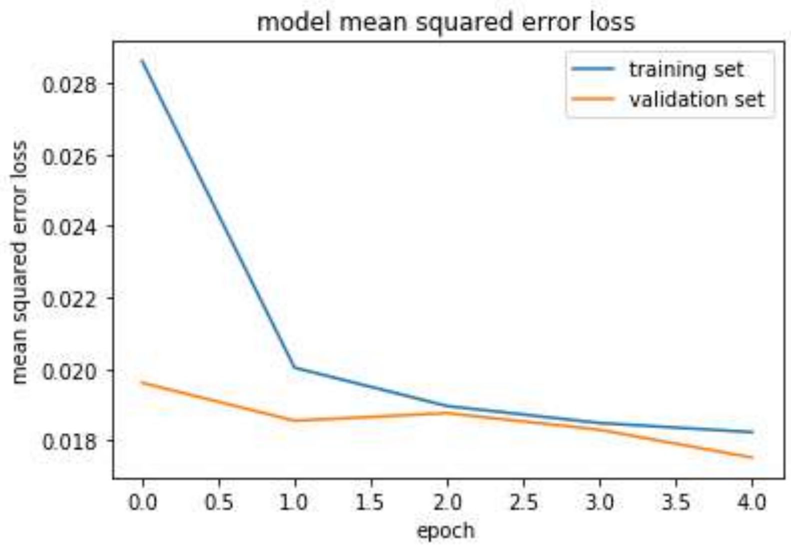
- Input layer with normalizing
- Convolutional layer
- Nonlinear layer
- Fully connected layer
- Maxpooling
- Dropout

Has an attempt been made to reduce overfitting of the model?

Train/validation/test splits have been used, and the model uses dropout layers or other methods to reduce overfitting.

I have experimented the dropout in different position of the model to test them in different position of code.

The Maxpooling layer with convolutional layer and Dropout with fully connected layer had the better result to prevent overfitting.



1st Resubmission

For splitting the training dataset into training and testing subset the train_test_split function of sklearn.model_selection has been used.

```
X_train, X_test , Y_train, Y_test = train_test_split(X_train, Y_train, test_size=0.2,
random_state=0)
```

And for using the test subset I hanged to the model.fit function as follows to use the test subset for validating the model:

```
history_object = model.fit(X_train, Y_train, validation_split = 0.2,shuffle = True, epochs = 5 ,
verbose=1 , batch_size=128 ,validation_data=(X_test, Y_test))
```

Have the model parameters been tuned appropriately?

Learning rate parameters are chosen with explanation, or an Adam optimizer is used.

I have used the Adam optimizer as input parameter of compile function.

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

These are the default parameters of the Adam optimizer:

```
lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False
```

lr: float >= 0. Learning rate.
beta_1: float, 0 < beta < 1. Generally close to 1.
beta_2: float, 0 < beta < 1. Generally close to 1.
epsilon: float >= 0. Fuzz factor. If None, defaults to K.epsilon().
decay: float >= 0. Learning rate decay over each update.
amsgrad: boolean. Whether to apply the AMSGrad variant of this algorithm from the paper "On the Convergence of Adam and Beyond".

The parameters that I have tuned is batch_size. The default value of the batch_size is 32 but I have tuned it to 128.

```
history_object = model.fit(X_train, Y_train, validation_split = 0.2,shuffle = True, epochs = 5 , verbose=1 , batch_size=128)
```

Is the training data chosen appropriately?

Training data has been chosen to induce the desired behavior in the simulation (i.e. keeping the car on the track).

For the training data the csv file and the captured images have been used. The training images, which I have used are the Udacity's captured data because I couldn't drive the car exactly in the middle of the road.

I have used the center, left and right images and their steering value. According to the Udacity learning material I should have considered 0.2 to adjust the steering measurement for the side camera images. In the following are the steps to read and load data to dataset.

```
images = []
measurements = []
```

In first step I have opened csv file with the open built-in function.

```
open('./img/driving_log.csv')
```

Second, I read the csv file with reader function of csv and assign the output to reader variable.

```
reader = csv.reader(csvfile)
```

Third, I used for loop and read all lines of csv file line by line and append each line to a list variable -- called lines -- to use for further purposes, which will be explained later.

```
    for line in reader:
        lines.append(line)
```

I extract four values from each row/line of csv file – name of center, left and right image and steering value. Since the left and right images need adjustment I have summed 0.2 to steering value or subtracted 0.2 from the steering value.

```
for line in lines1:
    for i in range(3):
        source_path = line[i]
        filename = source_path.split('\\')[-1]
        current_path = './img/round1/' + filename.strip()

        image = plt.imread(current_path)
        images.append(resize(convert_hsv(image)))
        if i == 0 :
            measurement = float(line[3])
        elif i==1 :
            measurement = float(line[3]) + 0.2
        elif i== 2:
            measurement = float(line[3]) - 0.2

        measurements.append(measurement)
print(current_path)
```

I have only used two different csv files. One of the is Udacity data. In this training data as mentioned in learning materials we have often left curves. I wanted to augment the training data, therefore I turned the car and drove in another direction to have CW curves.

One more thing that I have done is changing the color space technics from previous projects and spatial techniques to change the size of the images.

```
# The original shape of image is (160x320) and it will be resized to (16x32)
def resize(img):
    return cv2.resize(img[:,:,1],(32,16))
```

I have used the cv2.resize because of memory efficiency and if we look at performance, each epoch takes only 2 seconds.

```
Epoch 1/5
19677/19677 [==============================] - 2s 112us/step - loss: 0.0289 - val
_loss: 0.0246
```

Another code line that I have changed is in drive.py. I use the image size of 16 row and 32 columns, therefore I have resized the image which is passed to model.predict function.

```
image_array = np.asarray(image)
myimage = image_array[None, :, :, :]

#resize the image to my size 16 x 32
```

```
myimage = ( cv2.resize((cv2.cvtColor(myimage[0],
cv2.COLOR_RGB2HSV))[:,:,1],(32,16))).reshape(1,16,32,1)

steering_angle = float(model.predict(myimage, batch_size=1))
```

Architecture and Training Documentation

Is the solution design documented?

The README thoroughly discusses the approach taken for deriving and designing a model architecture fit for solving the given problem.

The model of this project has been developed with Keras and is composed of the following layers.

- Input layer with normalizing
- Convolutional layer
- Nonlinear layer
- Fully connected layer
- Maxpooling
- Dropout

Is the model architecture documented?

The README provides sufficient details of the characteristics and qualities of the architecture, such as the type of model used, the number of layers, the size of each layer. Visualizations emphasizing particular qualities of the architecture are encouraged.

In the following the layers of the model are listed via summary function. I have highlighted the summary with blue to write my explanations to each layer with black color.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lambda_2 (Lambda)            (None, 16, 32, 1)         0
```

The input image to deep neural network model has the shape (16 x 32 x 1) as explained before becau se of memory efficiency I have resized the images.

```
model.add(Lambda(lambda x: x/127.5 - 1.,input_shape=(16,32,1)))
```

```
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 30, 8)         80
```

The first convolutional layer with 3x3 kernel and 8 output filters.

```
model.add(Convolution2D(8, (3, 3), kernel_initializer='normal',padding='valid'))
```

```
_________________________________________________________________
activation_4 (Activation)    (None, 14, 30, 8)         0
```

Nonlinearity in the model.

```
model.add(Activation('relu'))
```

```
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 15, 8)          0
```

Prevent overfitting

```
model.add(MaxPooling2D((2,2),padding='valid'))
```

```
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 5, 13, 8)          584
```

The second convolutional layer with 3x3 kernel and 8 output filters.

```
model.add(Convolution2D(8, (3, 3) ,kernel_initializer='normal',padding='valid'))
```

```
_________________________________________________________________
activation_5 (Activation)    (None, 5, 13, 8)          0
```

Nonlinearity in the model.

```
model.add(Activation('relu'))
```

```
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 2, 6, 8)           0
```

Prevent overfitting

```
model.add(MaxPooling2D((2,2),padding='valid'))
```

```
_________________________________________________________________
dropout_2 (Dropout)          (None, 2, 6, 8)           0
```

Prevent overfitting

```
model.add(Dropout(0.2))
```

```
_________________________________________________________________
flatten_2 (Flatten)          (None, 96)                0
```

```
model.add(Flatten())
```

```
_________________________________________________________________
dense_3 (Dense)              (None, 50)                4850
```

The Fully connected layer

```
model.add(Dense(50))
```

```
_________________________________________________________________
activation_6 (Activation)    (None, 50)                0
```

Nonlinearity in the model.

```
model.add(Activation('relu'))
```

```
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 51
```

The Fully connected layer

```
model.add(Dense(1))
```

```
=================================================================
Total params: 5,565
Trainable params: 5,565
Non-trainable params: 0
```

For training the model I have used the Adam optimizer and it has been mentioned in the previous section the default learning rate is 0.001.

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

After training the model is trained with 5 epochs then it should be validated with 20% of the values of the dataset.

```
history_object = model.fit(X_train, Y_train, validation_split = 0.2,shuffle = True, epochs = 5 , verbose=1 , batch_size=128)
```

Finally, it can be saved for testing in simulator.

```
model.save('model.h5')
```

Is the creation of the training dataset and training process documented?

The README describes how the model was trained and what the characteristics of the dataset are. Information such as how the dataset was generated and examples of images from the dataset must be included.

Simulation

Is the car able to navigate correctly on test data?

No tire may leave the drivable portion of the track surface. The car may not pop up onto ledges or roll over any surfaces that would otherwise be considered unsafe (if humans were in the vehicle).

No tire leaves the drivable area.