Behavioral Cloning

Behavioral Cloning Project

The goals / steps of this project are the following:

- Use the simulator to collect data of good driving behavior or use Udacity provided data
- Build, a convolution neural network in Keras that predicts steering angles from images
- Train and validate the model with a training and validation set
- Test that the model successfully drives around track one without leaving the road
- Summarize the results with a written report

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Files Submitted & Code Quality

1. Submission includes all required files and can be used to run the simulator in autonomous mode

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 which has recorded session of one lap of autonomous driving

2. Submission includes functional code

Using the Udacity provided simulator and my drive.py file, the car can be driven autonomously around the track by executing

The preferred mode is : Screen Resolution as 800 x 600 and Graphics as Quality Fastest

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

3. Submission code is usable and readable

The model.py file contains the code for training and saving the convolution neural network. The file shows the pipeline I used for training and validating the model, and it contains comments to explain how the code works.

Note: To run the code make sure that path to data is correct. I used the standard 8K dataset provided by Udacity to train the model

Model Architecture and Training Strategy

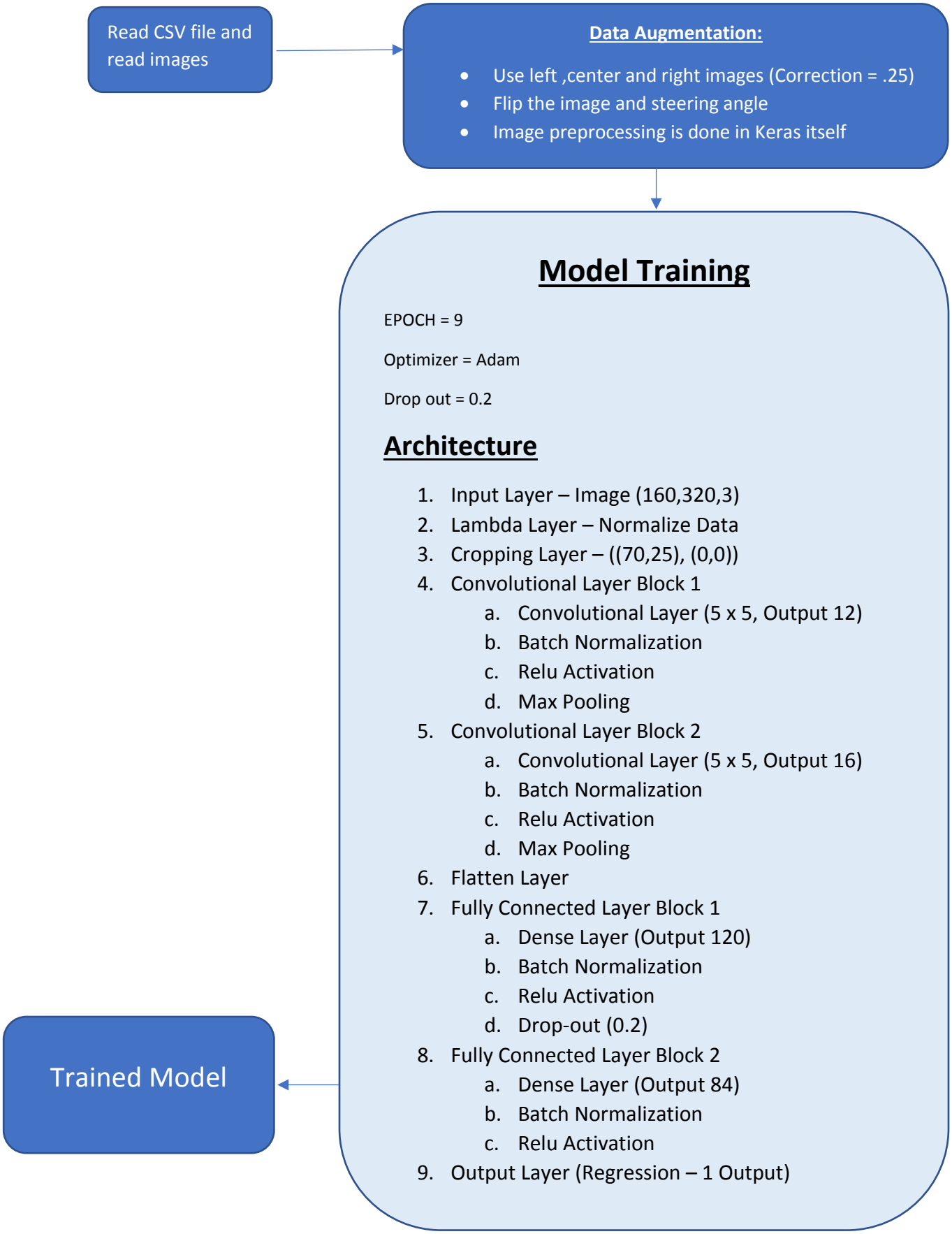
1. An appropriate model architecture has been employed

In model.py file you can find the code related to model architecture from line 70 to 112

I took the LeNet architecture and modified it to suit this problem. Following are changes done to the standard architecture.

- Image was prepressed in Keras itself. We normalized the images by dividing it by 255 and then subtracting 0.5 from it
- Image size is cropped by removing some upper (70 pixels) and lower (25 pixels) portion of the images. This was done to reduce training time and remove noisy data
- First convolutional layer has output size of 12 instead of 6 used in standard LeNet architecture
- Batch normalization is done between every linear and non-linear layer.
- Dropout (0.2) has been added to biggest fully connected layer to help generalize the model better
- Relu activation is used after every block of “conv + batch norm + max pool” layer and after fully connected layers

Flow Chart of the Pipeline including Model Architecture:



Please find below summary of the model structure:

Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 160, 320, 3)	0
cropping2d_1 (Cropping2D)	(None, 65, 320, 3)	0
conv2d_1 (Conv2D)	(None, 61, 316, 12)	900
batch_normalization_1 (Batch	(None, 61, 316, 12)	48
activation_1 (Activation)	(None, 61, 316, 12)	0
max_pooling2d_1 (MaxPooling2	(None, 30, 158, 12)	0
conv2d_2 (Conv2D)	(None, 26, 154, 16)	4800
batch_normalization_2 (Batch	(None, 26, 154, 16)	64
activation_2 (Activation)	(None, 26, 154, 16)	0
max_pooling2d_2 (MaxPooling2	(None, 13, 77, 16)	0
flatten_1 (Flatten)	(None, 16016)	0
dense_1 (Dense)	(None, 120)	1921920
batch_normalization_3 (Batch	(None, 120)	480
activation_3 (Activation)	(None, 120)	0
dropout_1 (Dropout)	(None, 120)	0
dense_2 (Dense)	(None, 84)	10080
batch_normalization_4 (Batch	(None, 84)	336
activation_4 (Activation)	(None, 84)	0
dense_3 (Dense)	(None, 1)	85

Total params: 1,938,713
Trainable params: 1,938,249
Non-trainable params: 464

2. Attempts to reduce overfitting in the model

Batch normalization was done in between every linear and non-linear layer. This helped the model a lot both in training and test results

Drop out (0.2) was used after the biggest fully connected layer, this helped in avoiding overfitting

Epoch of 12 was overfitting the model, so kept the epoch at 9

Did data augmentation so that the model doesn't overfit to some images

3. Model parameter tuning

Adam optimizer was used for training.

The correction factor for steering values when using left and right images had to be tuned. Tried 0.2, 0.22 and 0.25 values and found out 0.25 correction value performed better especially in the curves

Generator was used while training to reduce memory consumption. Tried batch size of 32 and 64 and 64 seem to work fine with the configuration I selected

Tried different epoch values, 12 tend to overfit the model and finalized 9 epochs

Tried different output size for convolutional layers (first one) and found 12 to be a good one. Therefore, changed the standard 6 output size of LeNet architecture to 12 for the first convolutional layer

Tried various drop out values and found 0.2 to be the best value

Tried the model with and without batch normalization and batch normalization worked nicely

4. Appropriate training data

I used the Udacity provided data (around 80K) and it was sufficient to train the model after data augmentation

Along with center image used the left and right images also. Used correction of 0.25 instead of default 0.2 to the steering angle when using left and right images

Created additional images by flipping it and steering angles. It helped to generalize the model better

Model Architecture and Training Strategy

1. Solution Design Approach

I started building a simple model using the Udacity provided dataset (80K) and normalized the data. It didn't work well. Here I used only center images

Later added additional images by flipping the center image and steering angle. This model worked better than previous model but nowhere good enough for submission.

Now used the LeNet architecture and experimented with different structure. Modified the LeNet architecture by making the first convolutional layer output as 12. This improved the model.

Did Batch Normalization along with cropping the image size (cropped 70 pixels from top and 25 from bottom part of the image) and this improved the model performance considerably. Also, this helped in model training

The model was not performing great in some curves and therefore added left and right images with correction of 0.25. Didn't go with default 0.2 correction, as 0.25 seem to make model work better in curves.

Added dropout on 0.2 to the biggest fully connected layer and this made the model work great.

Later played with number of epochs and found 5 was good enough when not using generator function, but when generator function was used I had to use 9 epochs.

2. Final Model Architecture

In model.py file you can find the code related to model architecture from line 70 to 112. More details about the model can be found in the section above. Please find below the model summary:

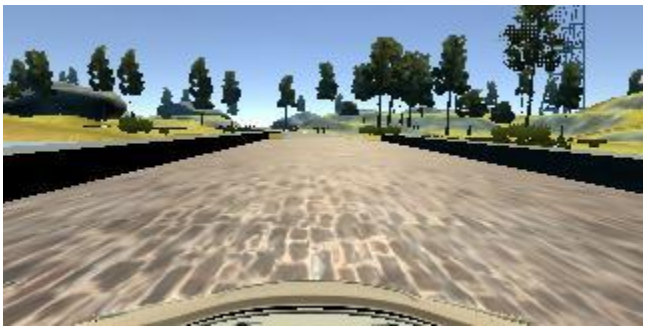
Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 160, 320, 3)	0
cropping2d_1 (Cropping2D)	(None, 65, 320, 3)	0
conv2d_1 (Conv2D)	(None, 61, 316, 12)	900
batch_normalization_1 (Batch	(None, 61, 316, 12)	48
activation_1 (Activation)	(None, 61, 316, 12)	0
max_pooling2d_1 (MaxPooling2	(None, 30, 158, 12)	0
conv2d_2 (Conv2D)	(None, 26, 154, 16)	4800
batch_normalization_2 (Batch	(None, 26, 154, 16)	64
activation_2 (Activation)	(None, 26, 154, 16)	0
max_pooling2d_2 (MaxPooling2	(None, 13, 77, 16)	0
flatten_1 (Flatten)	(None, 16016)	0
dense_1 (Dense)	(None, 120)	1921920
batch_normalization_3 (Batch	(None, 120)	480
activation_3 (Activation)	(None, 120)	0
dropout_1 (Dropout)	(None, 120)	0
dense_2 (Dense)	(None, 84)	10080
batch_normalization_4 (Batch	(None, 84)	336
activation_4 (Activation)	(None, 84)	0
dense_3 (Dense)	(None, 1)	85

Total params: 1,938,713
Trainable params: 1,938,249
Non-trainable params: 464

3. Creation of the Training Set & Training Process

I used the default dataset provided by Udacity (80K images) and with some data augmentation this data was enough to train the model.

Initially used only the center image for training the model.



Later the model was working good on straight road but not so great at curves, so used left and right images as well. While doing so I had to add a correction of 0.25 to steering angle to left image and subtract 0.25 from the steering angle for right image.

You can differentiate the images by looking at car hood position in the images

There were lot of data in images which were not required, like the top portion which usually contains images of trees and the bottom part of the image which has hood of the car. So, cropped the upper and lower part of the image to get rid of them. So that the data becomes less noisy and easy to train.

Added additional data by flipping the images and corresponding steering angle.

The 8K images became around 48K (6 times) after data augmentation and this was sufficient to train the network

Used generator function to feed the data during training and set aside 20% data for validation set.