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 video file generated from autonomous driving using my model.h5 file

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

```
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.

Model Architecture and Training Strategy

1. Model architecture

I used Nvidia model architecture (https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/) for this project.

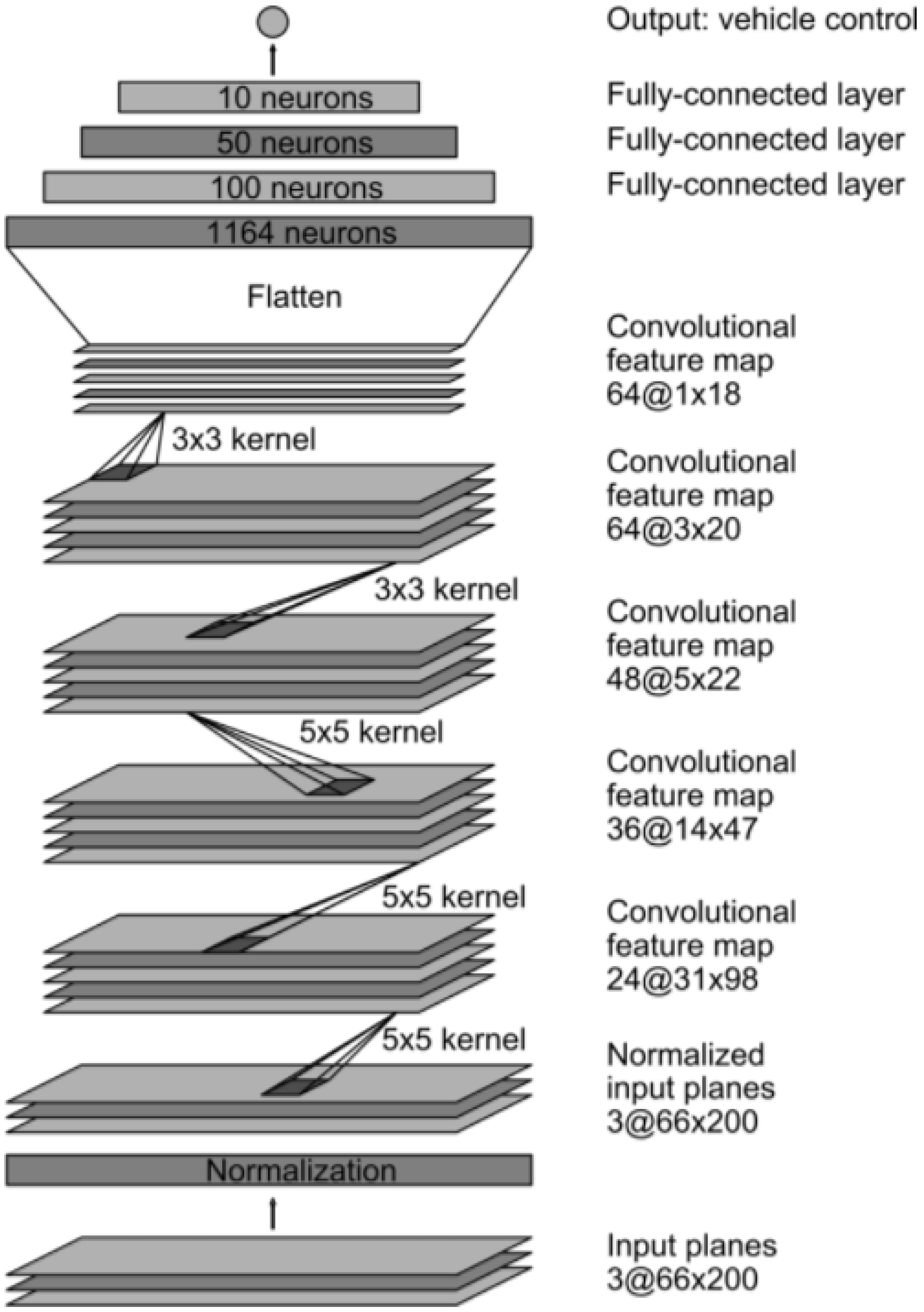

Figure 5: CNN architecture. The network has about 27 million connections and 250 thousand parameters.

Model is implemented in function *build_model()* which takes input shape of the image and the drop out probability to be used at the end of all convolution layers. The network architecture, consists of 9 layers, including a normalization layer, 5 convolutional layers, and 3 fully connected layers. The input image is used as RGB format itself whereas in original Nvidia model YUV format is used.

The model includes ELU(Exponential Linear Unit) layers to introduce nonlinearity, and the data is normalized in the model using a Keras lambda layer.

2. Attempts to reduce overfitting in the model

The model contains a single dropout layer at the end of all convolution layers in order to reduce overfitting.(model.py line 104)

The model was trained and validated on different data sets to ensure that the model was not overfitting. Train-test split of 80-20% of the total data is used (model.py function load_data() at line 167 and 26). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually.

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, recovering from the left and right sides of the road. Also added more training data from region where the car was getting off-track, by just repeating driving in particular region to collect more data.

For details about how I created the training data, see the next section.

Model Architecture and Training Strategy

1. Solution Design Approach

I used Nvidia model architecture directly since it was mentioned in one of the module that is a standard model being used by Nvidia and also from the old student Paul Heraty as mentioned in the Udacity module. To reduce overfitting I used a single dropout layer with probability of 0.6.

The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track. To improve the driving behavior in these cases, I collect more data by driving in this region repeatedly and also doing recovery driving around those problematic regions. In addition to this I drove one full lap of recovery driving to intentionally steer from left/right side of the road to center.

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

2. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded two laps on track one using center lane driving. I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to recover from sides of the road to drive back to center of the road.

To augment the data sat, I also flipped images and angles thinking that this would provide more data points for the model to train. For example, here is an image that has then been flipped:

After the collection process, I had 53913 number of data points. I then preprocessed this data by cropping to remove unnecessary parts of the image and resized it to match the input size of nvidia model. I augmented data to generate 3 additional image for each image,

1. Flipped images with opposite sign of steering angle
2. Randomly adjusted brightness image
3. Flipped image of randomly adjusted brightness image

In the end I had a total of 215652 data points.

I finally randomly shuffled the data set and put 20% of the data into a validation set.

I used this training data for training the model. The validation set helped determine if the model was over or under fitting. I used an adam optimizer so that manually training the learning rate wasn't necessary.