Behavioral Cloning Project
[![Udacity - Self-Driving Car NanoDegree](https://s3.amazonaws.com/udacity-sdc/github/shield-carnd.svg)](http://www.udacity.com/drive)

The goals / steps of this project are the following:
* Use the simulator to collect data of good driving behavior
* 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

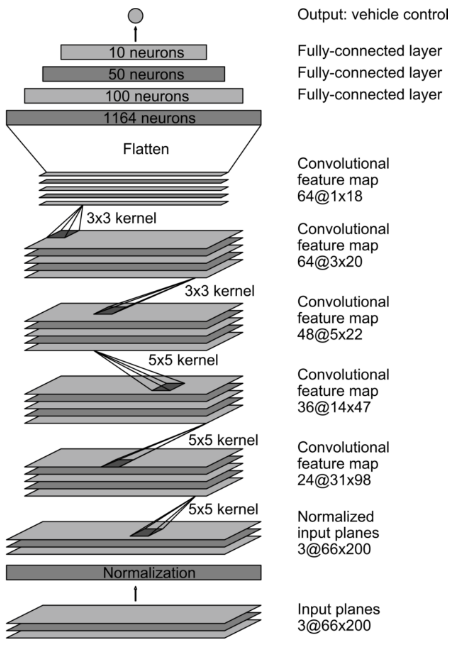
Rubric Points

Files Submitted & Code Quality
- `model.py`
- `model.h5`
- `output_video.mp4.mp4`
- `README.md`
- `drive.py`

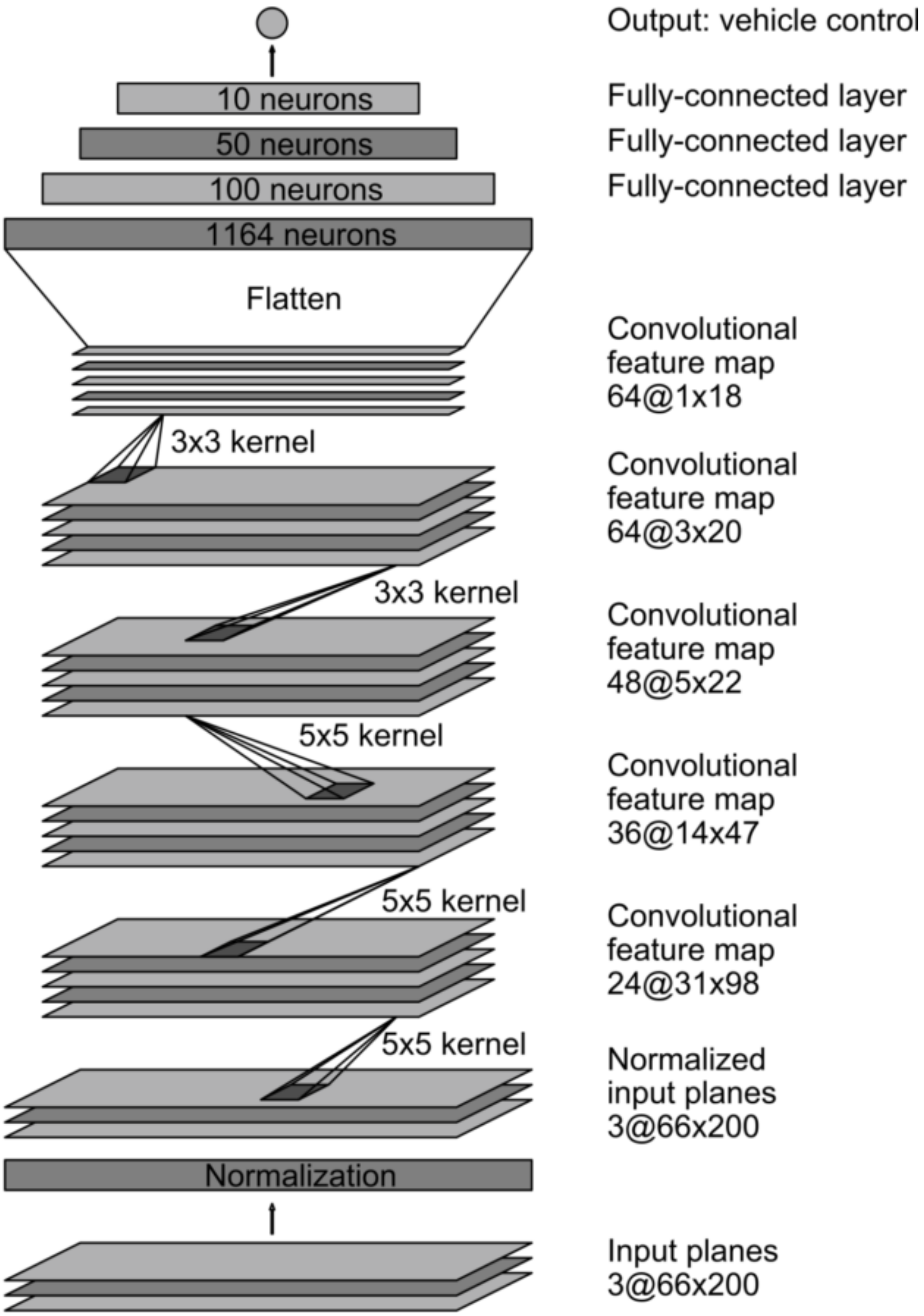
Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

For the actual network architecture, the Nvidia’s DAVE2 model is used because of its simplicity and demonstrated ability to perform well on self-driving car tasks. It consists of 5 Convolution layers followed by MaxPooling with a stride a of (1,1). It also consists of 5 fully connected layers. I am using MSE as loss function and Adam as gradient optimizer

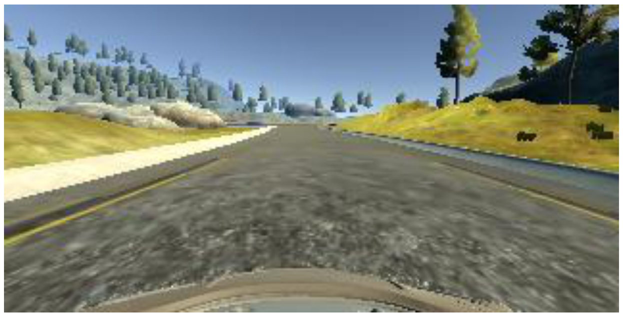


```
#### 2. Attempts to reduce overfitting in the model
To prevent overfitting, I used several data augmentation techniques like flipping images horizontally
as well as using left and right images to help the model generalize.
The model was trained and validated on different data sets to ensure that the model was not overfitting.
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 middle on the road. I used a combination of center lane driving, recovering from the left and right sides of the road.
For details about how I created the training data, see the next section.

### Model Architecture and Training Strategy

#### 1. Solution Design Approach

The overall strategy for deriving a good model was to use the [Nvidia](https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf) architecture since it has been proven to be very successful
in self-driving car tasks. I would say that this was the easiest part since a lot of other students have found it successful, the architecture was recommended in the lessons and it's adapted for this use case.

In order to gauge how well the model was working, I split my image and steering angle data into a training(80%) and validation set(20%).
I found that my first model had a low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was overfitting.
To combat the overfitting, I used several data augmentation techniques like flipping images horizontally.
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 so I gathered training data again and trained my second time with more data.
At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

#### 2. Final Model Architecture
The final model architecture code is shown above.
Here is a visualization of the architecture:
```


Output: vehicle control
10 neurons
Fully-connected layer
50 neurons
Fully-connected layer
100 neurons
Fully-connected layer
1164 neurons
Flatten
Convolutional feature map 64@1x18
3x3 kernel
Convolutional feature map 64@3x20
3x3 kernel
Convolutional feature map 48@5x22
5x5 kernel
Convolutional feature map 36@14x47
5x5 kernel
Convolutional feature map 24@31x98
5x5 kernel
Normalized input planes 3@66x200
Normalization
Input planes 3@66x200

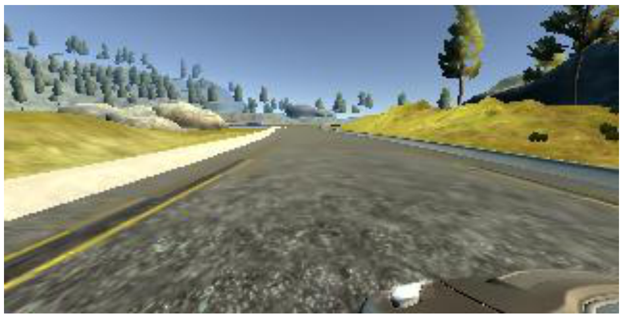
3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving:

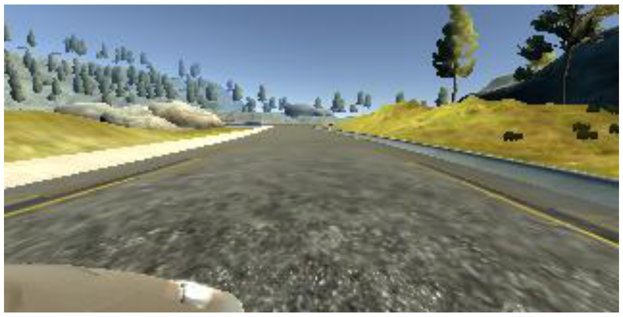


Center View

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 itself in case it deviates from its path.



Left Sided View



Right Sided View

To augment the data sat, I also flipped images and angles thinking that this would provide better symmetric view.

cropped image

flipped image

After the collection process, I had 18081 number of data points. I then preprocessed this data by merging all image paths in one list and putting angle measurements in another list.
I finally randomly shuffled the data set and put 20% of the data into a validation set and 80% in training set.
I used this training data for training the model. The validation set helped determine if the model was over or under fitting. The ideal number of epochs was 3 as evidenced while training.
I used the Adam optimizer so that manually training the learning rate wasn't necessary.