1. An appropriate model arcthiecture has been employed

I used NVIDIA's model .
It has 9 layers: 1 normalization, 5 convolutional and 3 fully connected.
I used 2x2 stride in the first 3 convolutional layers and 3x3 kernel without stride in the last two.

2. Attempts to reduce overfitting in the model

The model has 3 dropout layers in order to reduce overfitting.

3. Model parameter tuning

I used an adam optimizer . I tried to set the learning manually but that led to worse results, so I decided not to do that.

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

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

Model Architecture and Training Strategy

1. Solution Design Approach

My first step was to try with VGG and googlenet. GoogleNet was way too complicated so I decided to go forward with VGG but, the result were not good enough, so I took the suggestion from the class and go forward with NVIDIA. I saw their paper and I implemented my network based on that.
After I decided to stick with NVIDIA'S model I tried to modify a little bit and search for improveents. One thing that I did was to modifiy the input shape from 66x200 to 80x320 and adding one more convolutional layer. But that didn't pay off.
Also I tried to remove one fully connected layer, and also to play with the dropout layers(removing them, or changing the percentage), but again the results got worse,. I got low mean squared error on train set but high on validation set, so I decided to keep the droput layers as they were

I thought this model is appropriate for 2 reasons: it was tested for this type of a problem, and because other models failed when I tried to implement them.

I evaluated in 2 steps: first by looking the mean squared error, and if that was fine I would go on the simulator and evaluate it there.

The first obvious problem that I encountered was that the car didn't know how to recover. I solved that by providing test data of how to recover from getting to close the side of the road.
Also I noticed that in some cases, if I wait to long(in curves) to recover it would be to late, so I tried to fix those errors earlier. Basically teaching the car to stay as close as possible to the center.

2. Final Model Architecture

The final model architecture consisted of following layers and sizes

5 convolutional layers:

24 5x5 kernel, 2x2 stride

36 5x5 kernel 2x2 stride

48 5x5 kernel 2x2 stride

64 3x3 kernel

64 3x3 kernel

All layers (including the fully connected layers) had ELU as activation function.

3. Creation of the Training Set & Training Process

To capture good driving behaviour I used the sample data from the class, but I added about 4 laps and 1 lap in reverse, because the lap is so left oriented.
I also added data do recovery.

Here is a capture of the center camera

I trained my model to recover by taking the left and right side of the track and recovering towards the center the track:

I tried to add brightness and blur to the images but I didn't see any improvements with that so I decided not to do that.
As I said I didn't do any preprocessing, because of the bad results I've got.
I used a Keras lambda layer for normalization just as described in the class

I had a sample of 30415 and I used a 20% split for validation.

To test my model I used:

```
python drive.py model.json
```