Training Data
I found it almost impossible to generate training data from simulator using eith mouse or keyboard. I don't know about joystick since I had none. Therefore, I decided to stick to the sample training data provided by udacity. It seems to have worked well. I generate many more synthetic images from the provided ones to have enough training samples.

Model Architecture (model.py)
I experimented with few model architectures: BasicNet, LeNet, and settled or NvidiaNet as it performed best. There did not seem to be any problem of overfitting. The model uses subsampling of 2x2 after first 3 conv layers to reduce the overall number of parameters.

Layer Name	**details**
Cropping	Crop 50 pixels from top and 20 from bottom
Lambda	Normalization to bring intensity in the range -0.5 to 0.5
Conv2d + relu	24 filters of size 5x5, and subsample 2x2
Conv2d + relu	36 filters of size 5x5, and subsample 2x2
Conv2d + relu	48 filters of size 5x5, and subsample 2x2
Conv2d + relu	64 filters of size 3x3
Conv2d + relu	64 filters of size 3x3
Fully-connected1 + relu	100
Fully-connected2 + relu	50
Fully-connected3 + relu	10
Fully-connected4	1

Learning & Optimization (clone.py)
I experimented with RmsProp, Adam, and AdaDelta optimizers. AdaDelta gave the best results by far. On training with Adam, I have a car which was able to complete the full path, but steering was flickering a right and the car was continuously turning left and right, while maintaining on the given track.

The best learning rate was 0.001. Training for 25 epochs seems to work reasonably well. The performance wasn't good enough on training for 6 epochs. ALso, training and validation loss were decreasing so it made sense to train for more iterations. I used 20% of the training data as validation set.

Preprocessing (clone.py)
Using pre-processing certainly helped with improving performance. I used original 8000 data points and created total 40000 out of them. These are pre-processing techniques I used:

1. Used center, left, and right images
2. Used flipped images
3. Used randomly brightness modified images
4. Used images with randomly added shadow

I randomly chose one of the center, left, or right images to apply the pre-processing steps (3) and (4). Since, using too many variations of the same image would just lead to too much redundancy in training data. This might badly affect the generalization and accuracy of the model.

Exponential Smoothing (drive.py)
I noticed some flickering in the final drive of the car. This is imaginable since we are predicting steering for each time instance looking at just that image. In order to have smoothness in the predicted steering angle, I used “exponential-smoothing”. It resulted in slight improvement.

Although, a better approach would be to used LSTM layers as well which will result into using temporal information for predicting steering angle at any time instance.