Behavioral Cloning

Model:
I started out by watching the project video walkthrough by David Silver which turned out to be extremely helpful in getting started. I'm still new to python so things like reading and parsing the csv and image files still takes me a while and the walkthrough really sped that up. I followed along with the video and implemented the augmentation of the data set (flipping the images, normalizing), all pretty straightforward stuff already covered, and finally used the leNet architecture as a starting point. I also tried using the several other architectures, one from Nvidea's end-to-end self driving car program and another simpler architecture that had 3 Convolutional layers(32,32,64 filters), flattened layer and a fully connected layer.
I tested each of these with a variety of training data and what I found was that the Nvidea architecture showed real promise in the early part of the course but would always veer off-course at the sharper turns or run into the wall of the bridge or something like that. After experimenting with the training data, I finally figured out that the models were generally over-fitting so I went back down to the simpler CNN's and chose leNet after trial and error.

Training Data:
When I was using the Nvidea CNN, I kept trying to get the sweet spot with getting good turns, center driving, enough dirt tracks etc.. but the model would always fail somewhere. When it would fail on a curve, I would try to get more sharp curve training data. But then it would fail somewhere even sooner in the course, this made me think that it was overfitting. So I took the same training data which was pretty robust(~4 full laps, several sharp turns, several recoveries from the edge of the road) and went back to the leNet architecture and upped the epochs to 5 and that did the trick. On the previous Nvidea architecture, the car was going off on the bridge so I would tap the left and right keys. This model was hitting the wall so I thought this would be a way for the network to learn how to recover but it didn't end up helping. However I did get very different results using leNet. The leNet trained model ended up twitching left, right on the bridge so it truly cloned the behavior in that respect.
Before even training I did the standard data modification of normalizing and flipping the images and they ended up looking like this: