Akshay Khole
SDCND - Project 3
akshay@reversehack.in

Has an appropriate model architecture been employed for the task?

For Behavioural cloning we are using the NVIDIA model. The NVIDIA model has been tried and testing by NVIDIA and we have tried to apply that to this project. This is the model described in this paper:
https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf

Has an attempt been made to reduce overfitting of the model?

To reduce overfitting we are using the following techniques:

1. The input data has been split into Training and Validation sets. This keeps the validation set separate from the training set. The testing set would be input for Track2. But I am still experimenting with Track 2.
2. We use normalization as a pre-processing technique to extract important features from the image and train the model based on it.
3. I am also augmenting the data set by Flipping each image over the vertical axis and flipping value of the steering angle. I am also using images from the left and right cameras (and flipping these images and their respective steering angles). This helps to generalize the model. The model now sees normalized images of the track, it's curve and corresponding angles. We also crop each input image. We crop out 50 pixels from the top to remove the sky and crop off 20 pixels from the bottom to remove image of the hood of the car. Hence we are generalizing the model over input images. More techniques like brightness adjustment, small rotations can be applied to the image to further reduce overfitting and effectively train the model for general driving conditions.
4. I am using 7 Epochs because I trained the model a few times over 10 epochs and at 7 epochs the validation accuracy started dropping indicating that it was overfitting. So I trained the model for 7 epochs.
5. I am adding a Dropout layer of 50% dropout right before I flatten the output of all the convolutional layers. I figured after all the convolutions, dropping the higher level features may help the model generalize behaviour better using higher level features. The 50% value was randomly chosen and seems to work well.

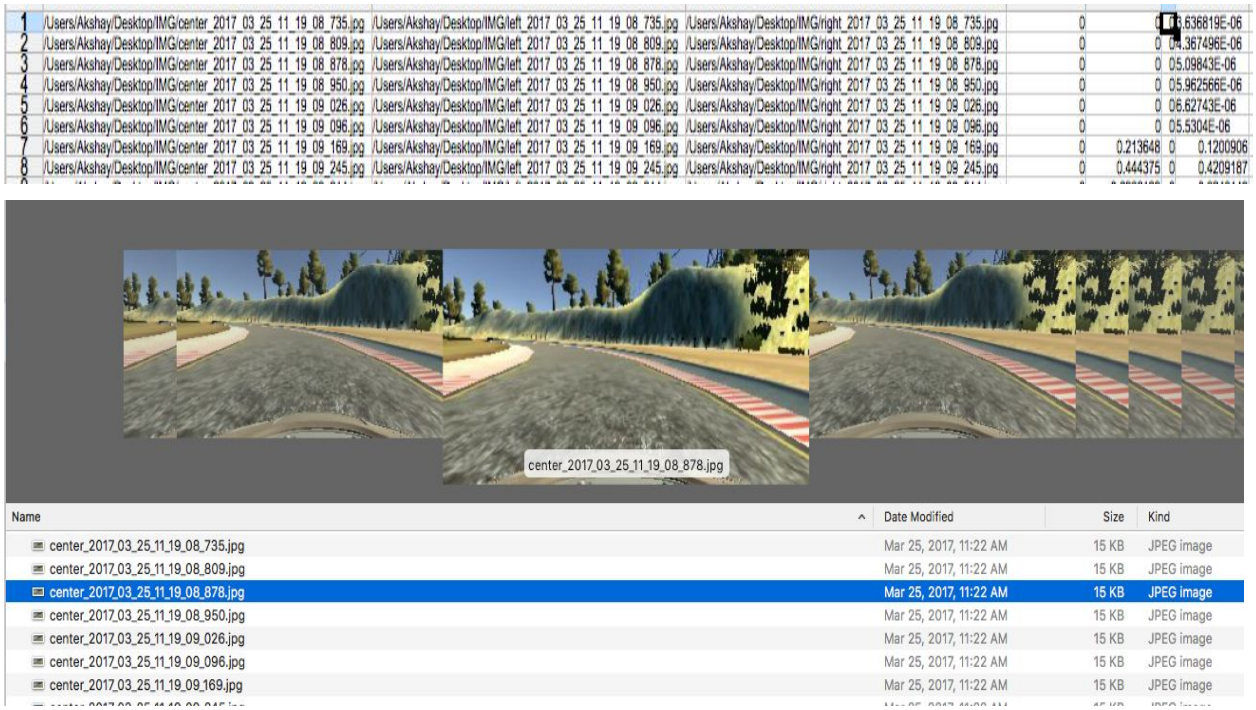
Have the model parameters been tuned appropriately?

To train the model we are using the Adam optimizer to tune to model for accuracy.

Is the training data chosen appropriately?

To train the model, we are using the data provided by Udacity. This dataset is a good training input. It contains good driving behaviour of a car driving safely and turning at curves to stay in the center of the lane. I could say this by inspecting random groups of images from the dataset. The images indicate the car stays mostly in the center of the lane and turns on curves.
If you notice the first few images, we can see that the car is fairly in the center of the lane and steering angles are 0 for all these images. Before training, I am manually removing the 1st row of

labels from the driving_log CSV. NVIDIA recommends using more training data with curved roads than straight roads. This ensures model does not overfit to straight driving conditions. Our track is mostly full of curves and very short straight driving parts. So our training data satisfies this recommendation.



Architecture and Training Documentation

We are using NVIDIA's model for our project (https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf).

I read the paper and understood the overall architecture of the model. The images below (taken from the paper) give a bird's eye view of the system and the training process:

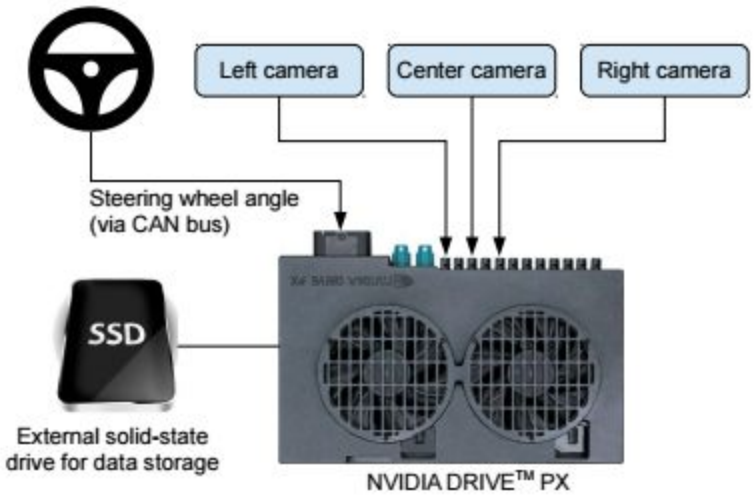


Figure 1: High-level view of the data collection system.

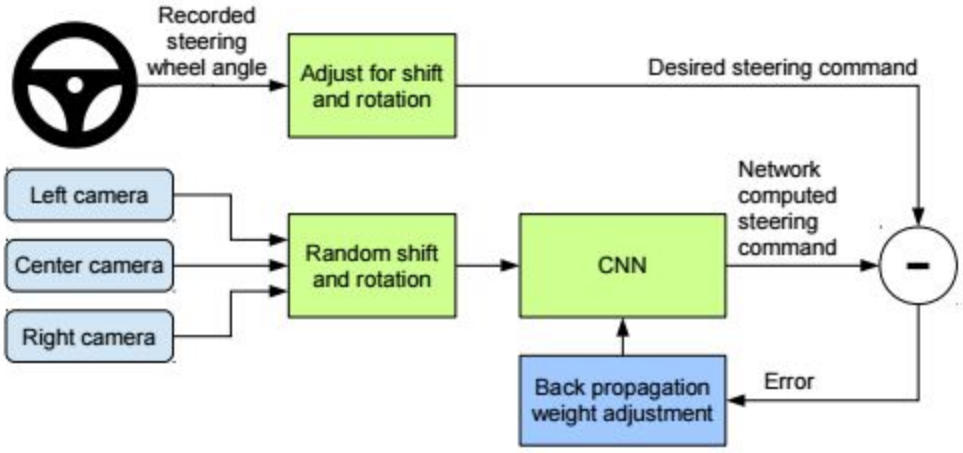


Figure 2: Training the neural network.

The architecture and sequence of the model and the layers is as follows:

1. Normalization layer
2. 5 Convolutional layers
3. Flattening layer
4. 3 Fully connected layers
5. Output

We begin with a Sequential layer and give it an input of Normalized images from our training set. We then add a Cropping layer that crops the top 50 and bottom 20 pixel blocks from the images.

We then add 5 convolutional layers one after the other. Each convolutional layer has a stride of (2, 2) and we use a (5, 5) kernel.

We then flatten our output from the last convolutional layer and feed it to the last 3 fully connected layers. The output from the last fully connected layer gives us the steering angle of the car.

Layers and Dimensions:

Input Shape - 160 x 320
1st Conv2D with 24 outputs
2nd Conv2D with 36 outputs
3rd Conv2D with 48 outputs
4th Conv2D with 64 outputs
5th Conv2D with 64 outputs
50 % Dropout Layer - added by me to remove overfitting (not part of nvidia arch)
Flattening layer
1st Fully connected layer with 100 outputs
2nd Fully connected layer with 50 outputs
3rd Fully connected layer with 1 output

Non-linearity:

We introduce non-linearity by adding a RELU activation layer with each Convolutional layer.

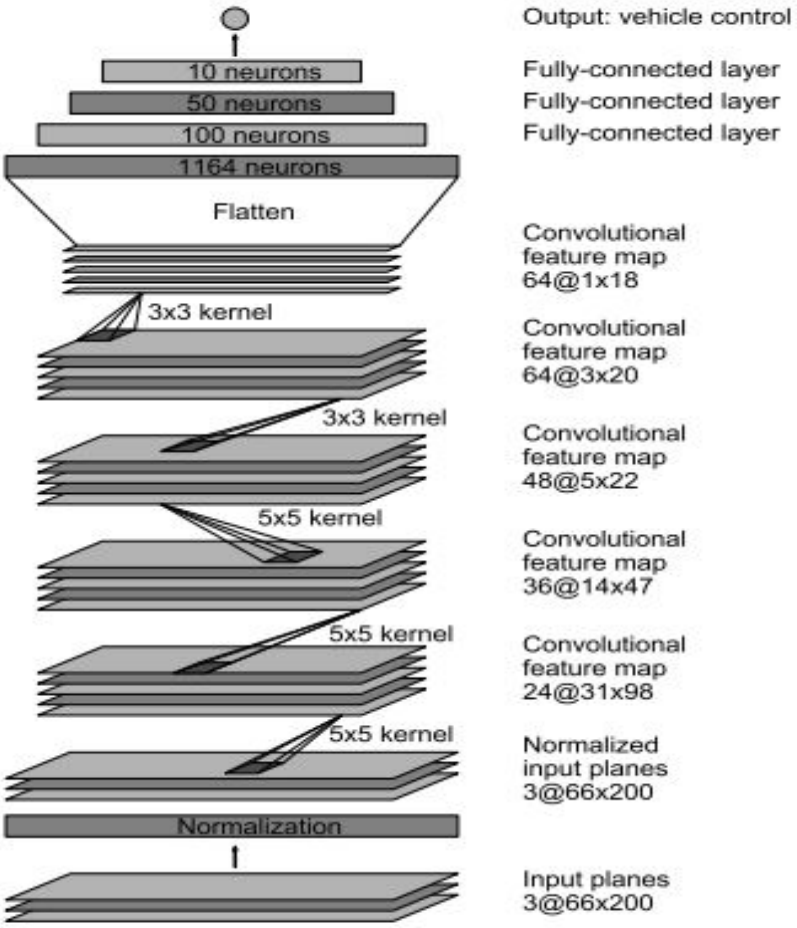


Figure 4: CNN architecture. The network has about 27 million connections and 250 thousand parameters.

Architecture of the NVIDIA model (taken from paper)

How the model was trained:

To train the model we used a generator to provide batches (of size 32) to the model to train on. This keeps the memory utilization low and ensures training happens over larger data sets. We then save the model to a file (model.h5). I am using 7 Epochs because I trained the model a few times over 10 epochs and at 7 epochs the validation accuracy started dropping indicating that it was overfitting. So I trained the model for 7 epochs.

Is the car able to navigate correctly on test data?

Yes the car is able to drive by itself on test track 1. The link to the video of car driving on test track 1 is available here:
https://drive.google.com/file/d/0BykAMon4p-hyVWU0VEx1cFFBbkk/view?usp=sharing

It was created using QuickTime player’s screen record on a Macbook and should be playable using Quicktime.