Self-Driving Car Game

Team Members

Aswathnarayan Muthukrishnan Kirubakaran

Archana Chintha

Project Description

In this project, we use a neural network to clone car driving behavior. It is a supervised regression problem between the car steering angles and the road images in front of a car.

Those images were taken from three different camera angles (from the center, the left and the right of the car).

The network is based on The NVIDIA model, which has been proven to work in this problem domain.

As image processing is involved, the model is using convolutional layers for automated feature engineering.

Files included

- model.py The script used to create and train the model.
- drive.py The script to drive the car. You can feel free to resubmit the original drive.py or make modifications and submit your modified version.
- utils.py The script to provide useful functionalities (i.e. image preprocessing and augmentation)

Model Architecture Design

The design of the network is based on the NVIDIA model, which has been used by NVIDIA for the end-to-end self-driving test. As such, it is well suited for the project.

It is a deep convolution network which works well with supervised image classification / regression problems. As the NVIDIA model is well documented, we were able to focus how to adjust the training images to produce the best result with some adjustments to the model to avoid over fitting and adding non-linearity to improve the prediction.

We've added the following adjustments to the model.

- We used Lambda layer to normalized input images to avoid saturation and make gradients work better.
- We've added an additional dropout layer to avoid overfitting after the convolution layers.
- We've also included ReLu for activation function for every layer except for the output layer to introduce non-linearity.

In the end, the model looks like as follows:

- Image normalization
- Convolution: 5x5, filter: 24, strides: 2x2, activation: ReLu
- Convolution: 5x5, filter: 36, strides: 2x2, activation: ReLu
- Convolution: 5x5, filter: 48, strides: 2x2, activation: ReLu
- Convolution: 3x3, filter: 64, strides: 1x1, activation: ReLu
- Convolution: 3x3, filter: 64, strides: 1x1, activation: ReLu
- Drop out (0.4)
- Fully connected: neurons: 100, activation: ReLu
- Fully connected: neurons: 50, activation: ReLu
- Fully connected: neurons: 10, activation: ReLu
- Fully connected: neurons: 1 (output)

As per the NVIDIA model, the convolution layers are meant to handle feature engineering and the fully connected layer for predicting the steering angle. However, as stated in the NVIDIA document, it is not clear where to draw such a clear distinction. Overall, the model is very functional to clone the given steering behavior.

The below is a model structure output from the Keras which gives more details on the shapes and the number of parameters.

Layer (type)	Output Shape	Params	Connected to
lambda_1 (Lambda)	(None, 66, 200, 3)	0	lambda_input_1
convolution2d_1 (Convolution2D)	(None, 31, 98, 24)	1824	lambda_1
convolution2d_2 (Convolution2D)	(None, 14, 47, 36)	21636	convolution2d_1
convolution2d_3 (Convolution2D)	(None, 5, 22, 48)	43248	convolution2d_2

Layer (type)	Output Shape	Params	Connected to
convolution2d_4 (Convolution2D)	(None, 3, 20, 64)	27712	convolution2d_3
convolution2d_5 (Convolution2D)	(None, 1, 18, 64)	36928	convolution2d_4
dropout_1 (Dropout)	(None, 1, 18, 64)	0	convolution2d_5
flatten_1 (Flatten)	(None, 1152)	0	dropout_1
dense_1 (Dense)	(None, 100)	115300	flatten_1
dense_2 (Dense)	(None, 50)	5050	dense_1
dense_3 (Dense)	(None, 10)	510	dense_2
dense_4 (Dense)	(None, 1)	11	dense_3
	Total params	252219	

How to Make a Self-Driving Car

Udacity recently open sourced their self-driving car simulator originally built for SDND students

- built in Unity (free game making engine https://unity3d.com/)
- add new tracks, change prebuilt scripts like gravity acceleration easily

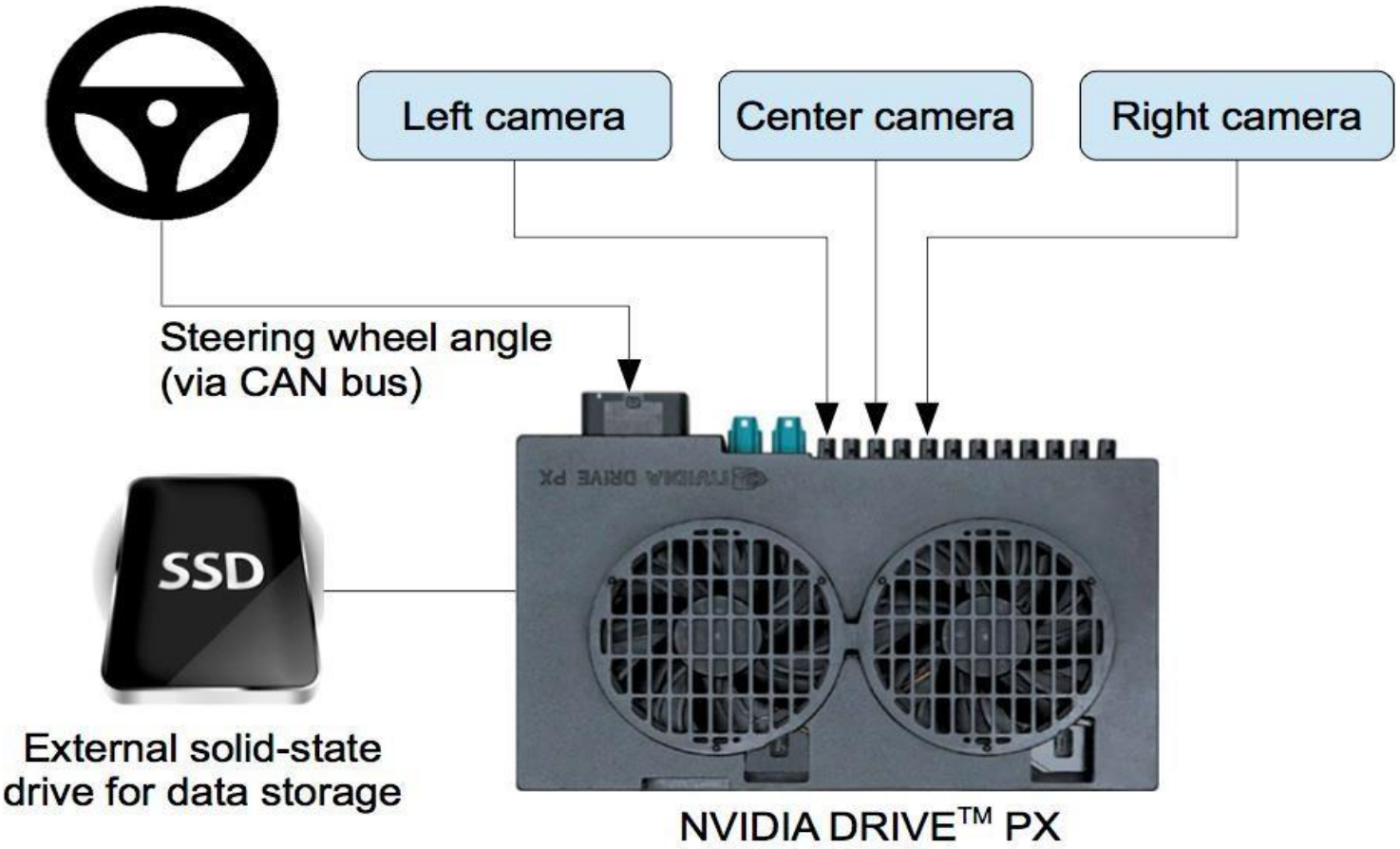
Data Generation

- Records images from center, left, and right cameras w/ associated steering angle, speed, throttle and brake.
- saves to CSV
- ideally you have a joystick, but keyboard works too

Training Mode – Behavioral cloning

We use a 9-layer convolutional network, based off of Nvidia's end-to-end learning for self-driving car paper. 72 hours of driving data was collected in all sorts of conditions from human drivers

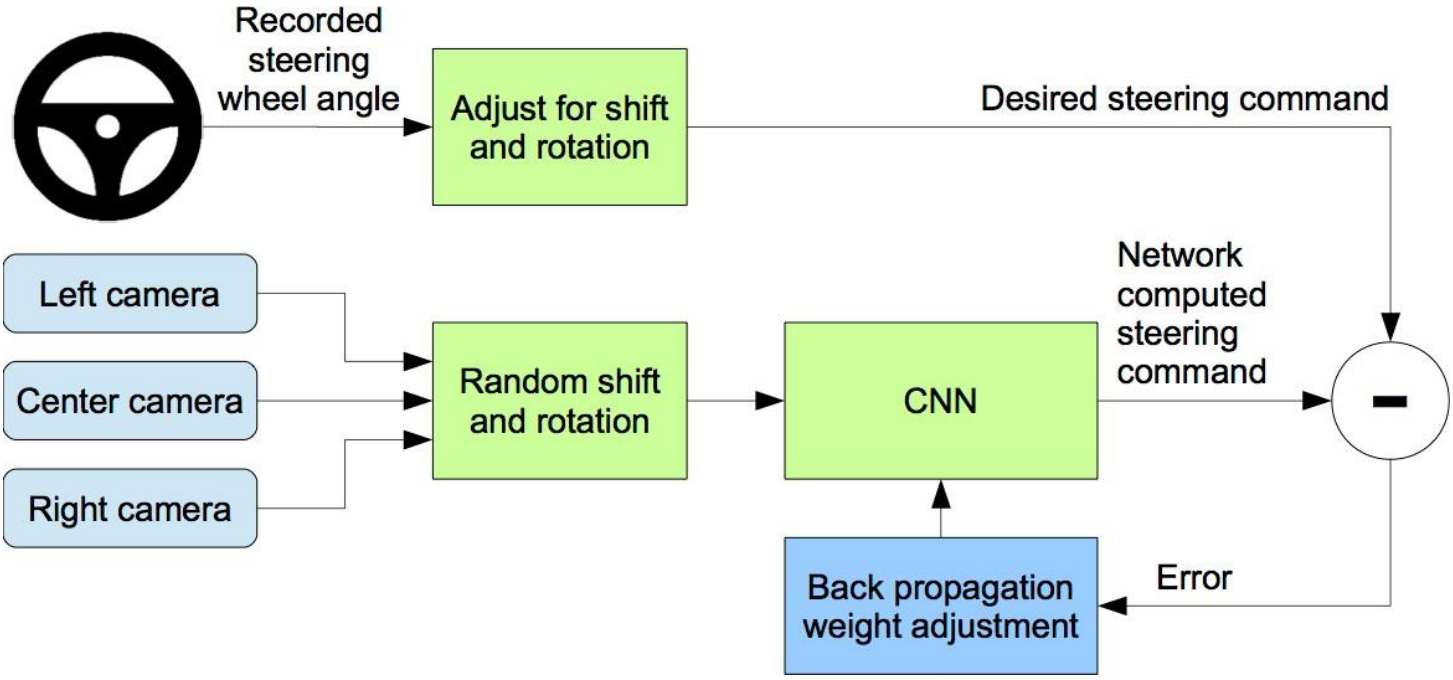
Hardware design:



- 3 cameras
- The steering command is obtained by tapping into the vehicle’s Controller Area Network (CAN) bus.
- Nvidia's Drive PX onboard computer with GPUs

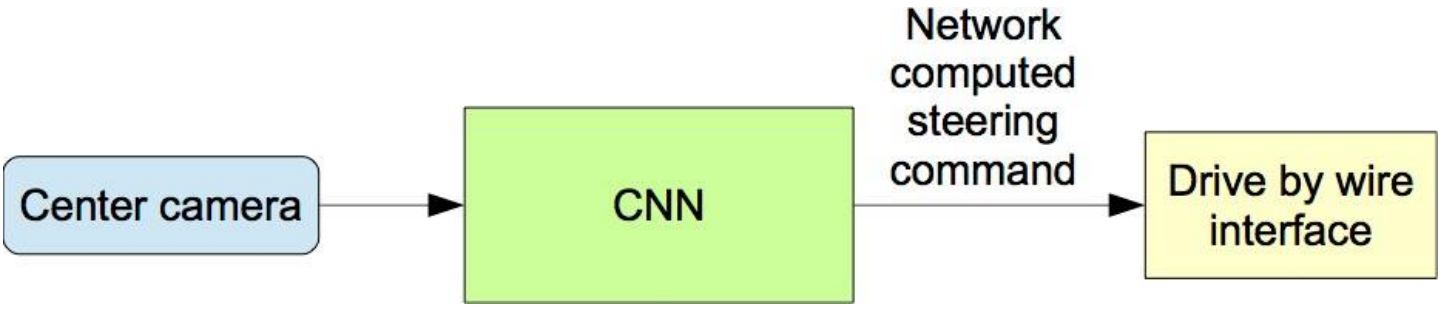
To make the system independent of the car geometry, the steering command is 1/r, where r is the turning radius in meters. 1/r was used instead of r to prevent a singularity when driving straight (the turning radius for driving straight is infinity). 1/r smoothly transitions through zero from left turns (negative values) to right turns (positive values).

Software Design (supervised learning) :¶



Images are fed into a CNN that then computes a proposed steering command. The proposed command is compared to the desired command for that image, and the weights of the CNN are adjusted to bring the CNN output closer to the desired output. The weight adjustment is accomplished using back propagation

Eventually, it generated steering commands using just a single camera



Data Preprocessing

Image Sizing

- the images are cropped so that the model won't be trained with the sky and the car front parts
- the images are resized to 66x200 (3 YUV channels) as per NVIDIA model
- the images are normalized (image data divided by 127.5 and subtracted 1.0). As stated in the Model Architecture section, this is to avoid saturation and make gradients work better)

Model Training

Image Augmentation

For training, we used the following augmentation technique along with Python generator to generate unlimited number of images:

- Randomly choose right, left or center images.
- For left image, steering angle is adjusted by +0.2
- For right image, steering angle is adjusted by -0.2
- Randomly flip image left/right
- Randomly translate image horizontally with steering angle adjustment (0.002 per pixel shift)
- Randomly translate image vertically
- Randomly added shadows
- Randomly altering image brightness (lighter or darker)

Using the left/right images is useful to train the recovery driving scenario. The horizontal translation is useful for difficult curve handling (i.e. the one after the bridge).

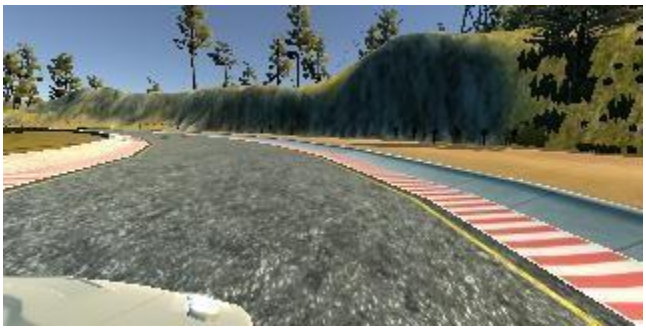
Examples of Augmented Images

The following is the example transformations:

Center Image

Left Image

Right Image



Flipped Image

Translated Image

Training, Validation and Test

We divided the images into train and validation set in order to measure the performance at every epoch. Testing was done using the simulator.

As for training,

- We used mean squared error for the loss function to measure how close the model predicts to the given steering angle for each image.
- We used Adam optimizer for optimization with learning rate of 1.0e-4 which is smaller than the default of 1.0e-3. The default value was too big and made the validation loss stop improving too soon.
- We used Model Checkpoint from Keras to save the model only if the validation loss is improved which is checked for every epoch.

The Lake Side Track

As there can be unlimited number of images augmented, we set the samples per epoch to 20,000. We tried from 1 to 200 epochs but we found 5-10 epochs is good enough to produce a well-trained model for the lake side track. The batch size of 40 was chosen as that is the maximum size which does not cause out of memory error on my Mac with NVIDIA GeForce GT 650M 1024 MB.

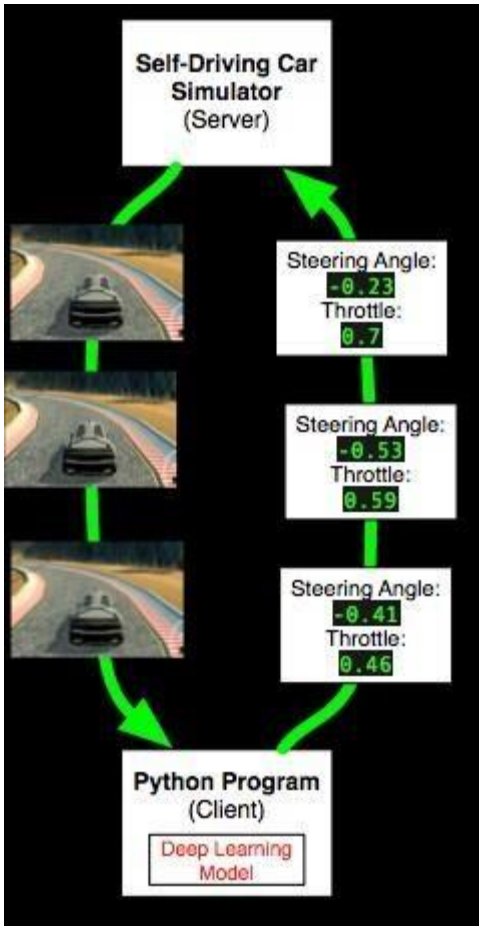
The Mountain Track

It's much more difficult than the lake side track.

We used the simulator to generate training data by doing 3 to 4 rounds. Also, added several recovery scenarios to handle tricky curves and slopes.

Testing mode

We will just run autonomous mode, then run our model and the car will start driving



The model can drive the course without bumping into the side ways.

The link below is the output which we automated the drive. We published the link in

YouTube https://www.youtube.com/watch?v=-zWhgLDXqyY

Acknowledgment

We would like to show our gratitude to professor Nik Bear Brown for guiding us during this project.

References

- NVIDIA model: https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/
- General Video Game AI: Learning from Screen Capture by Kamolwan Kunanusont, Simon M. Lucas, Diego Perez-Li ´ ebana https://arxiv.org/pdf/1704.06945.pdf.
- https://www.tensorflow.org/tutorials/wide_and_deep
- https://cognitiveclass.ai/courses/deep-learning-tensorflow/