CSYE 7374: Cognitive Computing and deep learning

Midterm Project: Self-driving cars using deep learning

Author: Dhanshri More

**Introduction:**

In a new automotive application, we have used convolutional neural networks (CNNs) to map the raw pixels from a front-facing camera to the steering commands for a self-driving car. This powerful end-to-end approach means that with minimum training data from humans, the system learns to steer, with or without lane markings, on both local roads and highways. The system can also operate in areas with unclear visual guidance such as parking lots or unpaved roads.

CNN have revolutionized pattern recognition. This project aim CNNs to go beyond pattern recognition and learn entire processing pipeline needed to steer an automobile.

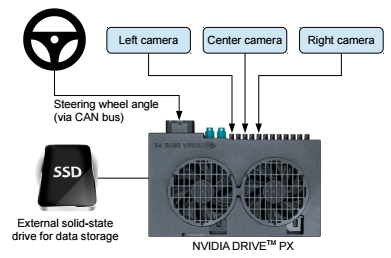
By letting the car figure out how to interpret images on its own, we can skip a lot of the complexity that exists in manually selecting features to detect, and drastically reduce the cost required to get an autonomous vehicle on the road by avoiding LiDAR-based solutions.

**Problem Statement:**

To train convolutional neural network (CNN) for self-driving cars that maps raw pixels from a single front-facing camera directly to steering commands for self-driving cars

**Dataset:**

In this project I have used image frames from a camera mounted to the windshield of our car, and predict the appropriate steering angle using convolutional neural networks and deep learning.



Dataset was in Robot operating system format ie: ROSbag format

Steps to get data in jpg format:

1. Download VirtualBox
2. Install Ubuntu 16.04 on VirtualBox
3. Install Docker with this link: <https://docs.docker.com/engine/installation/linux/ubuntulinux/>
4. Download rwightman’s repository to VirtualBox
5. Download transmission for torrents: [http://transmissionbt.com](http://transmissionbt.com/)
6. Download the torrents to your local machine
7. Setup shared directories between your VirtualBox and local machine/external storage
8. Run rwightman’s reader

Processed data can be downloaded at <https://drive.google.com/file/d/0B-KJCaaF7elleG1RbzVPZWV4Tlk/view>

**Sample data:**

Data.txt has information about image name and driving wheel angle at that time .

**Data processing:**

1. Read steering wheel angle in radians

#read data.txt

with open("driving\_dataset/data.txt") as f:

for line in f:

xs.append("driving\_dataset/" + line.split()[0])

ys.append(float(line.split()[1]) \* scipy.pi / 180)

1. Divide data in train and test set

xs\_train = xs[:int(len(xs) \* 0.8)]

ys\_train = ys[:int(len(xs) \* 0.8)]

xs\_test = xs[-int(len(xs) \* 0.2):]

ys\_test = ys[-int(len(xs) \* 0.2):]

1. Write function for batch processing

def LoadTrainBatch(batch\_size):

    global train\_batch\_pointer

    x\_out = []

    y\_out = []

    for i in range(0, batch\_size):

x\_out.append(scipy.misc.imresize(scipy.misc.imread(train\_xs[(train\_batch\_pointer + i) % num\_train\_images])[-150:], [60, 80]) / 255.0)

        y\_out.append([train\_ys[(train\_batch\_pointer + i) % num\_train\_images]])

    train\_batch\_pointer += batch\_size

    return x\_out, y\_out

def LoadValBatch(batch\_size):

    global val\_batch\_pointer

    x\_out = []

    y\_out = []

    for i in range(0, batch\_size):

 x\_out.append(scipy.misc.imresize(scipy.misc.imread(val\_xs[(val\_batch\_pointer + i) % num\_val\_images])[-150:], [60, 80]) / 255.0)

        y\_out.append([val\_ys[(val\_batch\_pointer + i) % num\_val\_images]])

    val\_batch\_pointer += batch\_size

    return x\_out, y\_out

**Infrastructure and Framework:**

**AWS EC2: g2.2xlarge**

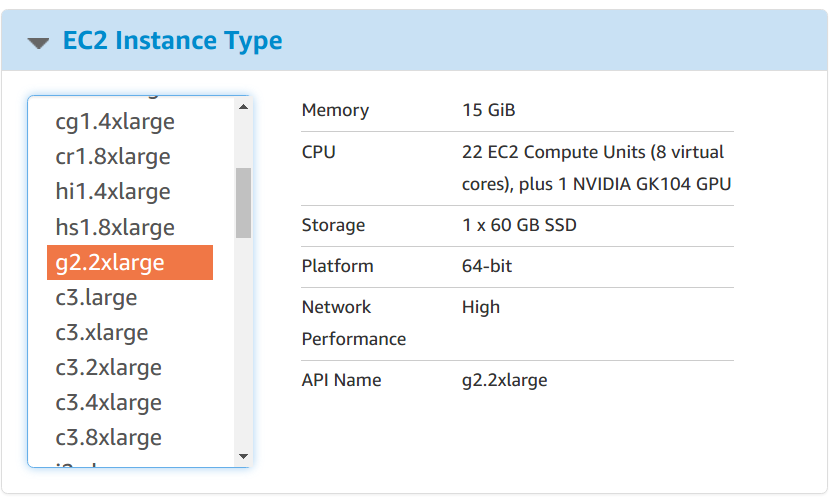


**Product Description**

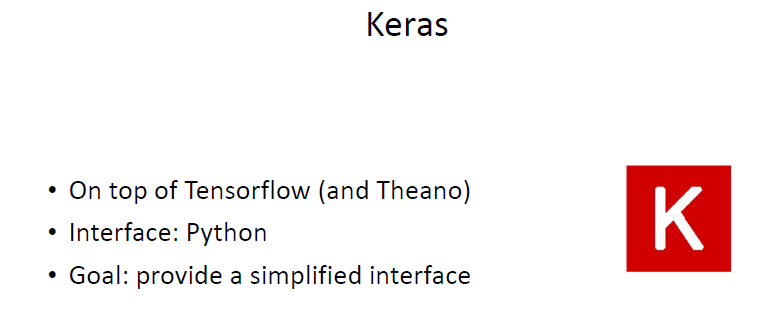
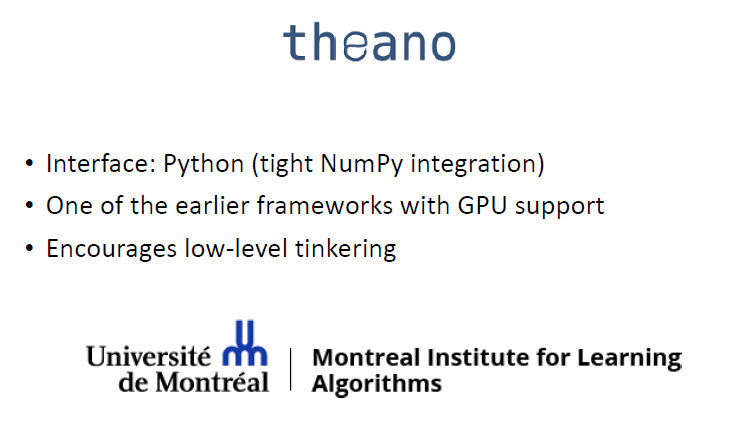
The Deep Learning AMI is an Amazon Linux image supported and maintained by Amazon Web Services for use on Amazon Elastic Compute Cloud (Amazon EC2). It is designed to provide a stable, secure, and high performance execution environment for deep learning applications running on Amazon EC2. It includes popular deep learning frameworks, including MXNet, Caffe, Tensorflow, Theano, CNTK and Torch as well as packages that enable easy integration with AWS, including launch configuration tools and many popular AWS libraries and tools. It also includes the Anaconda Data Science Platform for Python2 and Python3. Amazon Web Services provides ongoing security and maintenance updates to all instances running the Amazon Linux AMI. The Deep Learning AMI is provided at no additional charge to Amazon EC2 users.

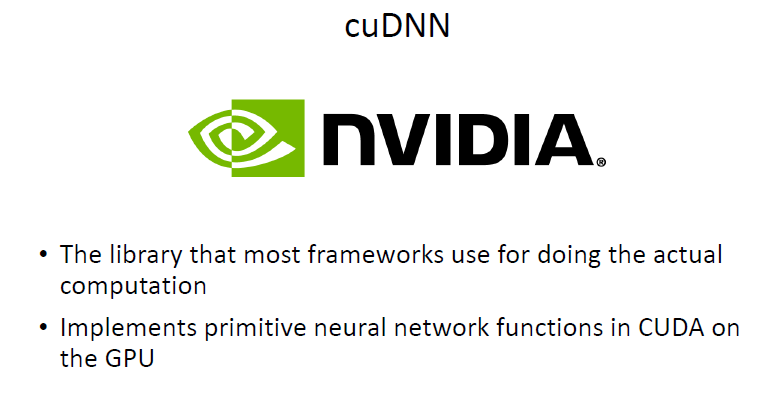
The AMI Ids for the Deep Learning Amazon Linux AMI are the following:  
us-east-1 : ami-e7c96af1  
us-west-2: ami-dfb13ebf

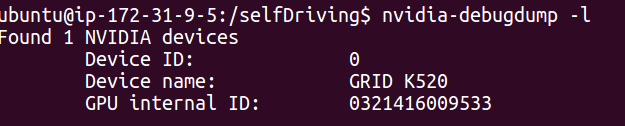
Release tags/Branches used for the DW Frameworks:  
MXNet : v0.9.3 tag  
Tensorflow : v1.0.0 tag  
Theano : rel-0.8.2 tag  
Caffe : rc5 tag  
CNTK : v2.0beta12.0 tag  
Keras : 1.2.2 tag



**Deep Learning Libraries and NVIDIA GPU:**







**Modeling:**

**Convolutional neural network:**

Trained convolutional neural net to process visual data from front camera mounted on vehicle. Perform hyper parameter tuning to get best model that gives steering angle for self -driving car

CNN have revolutionized pattern recognition. This project aim CNNs to go beyond pattern recognition and learn entire processing pipeline needed to steer an automobile.

**Model Architecture:**

model = Sequential()

model.add(Convolution2D(24, 5, 5, input\_shape=(66, 200,3), border\_mode='same', activation='relu'))

model.add(Dropout(0.5))

model.add(Convolution2D(36, 5, 5, border\_mode='same', activation='relu'))

model.add(Dropout(0.5))

model.add(Convolution2D(48, 5, 5, border\_mode='same', activation='relu'))

model.add(Dropout(0.5))

model.add(Convolution2D(64, 3, 3, activation='relu', border\_mode='same'))

model.add(Dropout(0.5))

model.add(Convolution2D(64, 3, 3, activation='relu', border\_mode='same'))

model.add(Dropout(0.5))

model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Flatten())

model.add(Dense(1024, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(100, activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(10, activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(1))

model.compile(optimizer=Adam(lr=1e-4), loss = 'mse')

**Optimizer: Adam**

I have used Adam as optimizer, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning.

Empirical results demonstrate that in this project, Adam with learning rate lr=1e-4 works well in practice and compares favorably to other stochastic optimization methods.

**Activation function:** ReLU

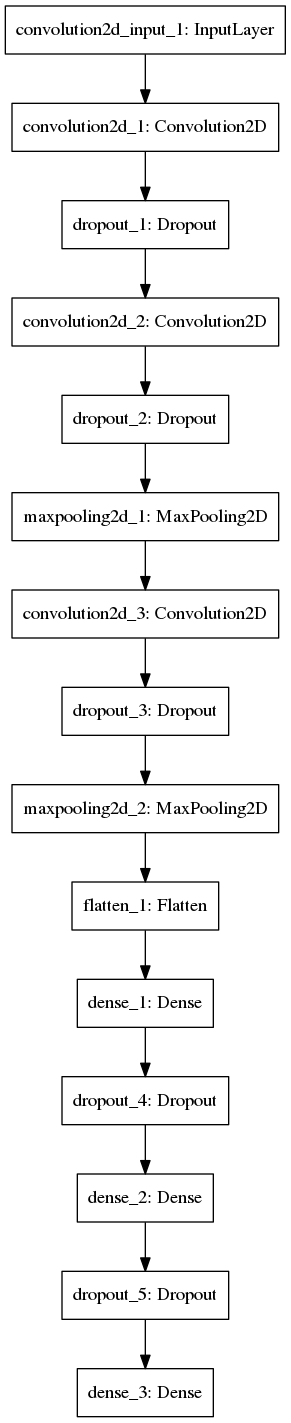
The advantages of using Rectified Linear Units in neural networks are

* If Hard max function is used as activation function, it induces the sparsity in the hidden units.
* ReLU doesn't face gradient vanishing problem as with sigmoid and tanh function. Also, It has been shown that deep networks can be trained efficiently using ReLU even without pre-training.

**Loss Function: MSE**

A loss function or objective function, or optimization score function is one of the two parameters required to compile a model

**MSE** ie Mean Squared error measures the [average](https://en.wikipedia.org/wiki/Expected_value) of the squares of the [errors](https://en.wikipedia.org/wiki/Error_(statistics)) or [deviations](https://en.wikipedia.org/wiki/Deviation_(statistics)). It is the difference between the estimator and what is estimated. MSE is a [risk function](https://en.wikipedia.org/wiki/Risk_function), corresponding to the [expected value](https://en.wikipedia.org/wiki/Expected_value) of the squared error loss



**Training:**

The network was initially trained on the small set of training data with no pre-processing on local Ubuntu system with 8GB RAM, but model has completed only 2 epoch in 3 hour on local Ubuntu machine. When tried to train same model on whole data set with input image as 66\*200, it gives Resource exhausted error. Amazon web services platform has been used to solve this issue

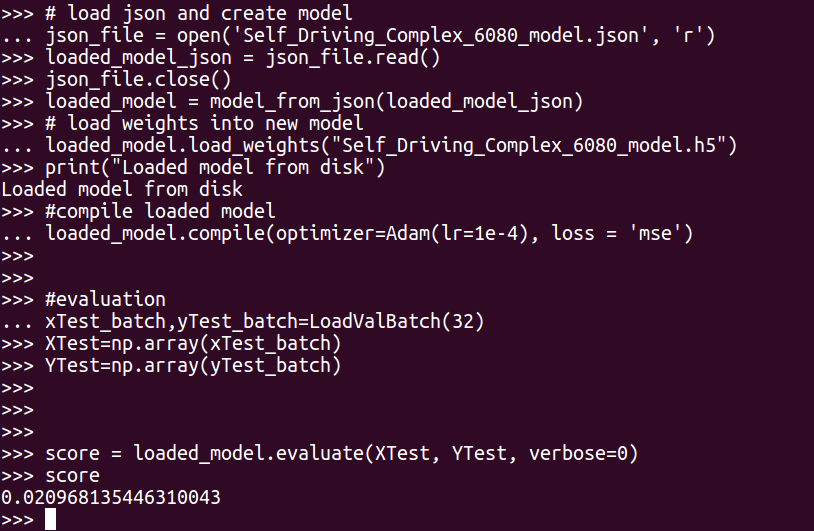
Model was then trained on Amazon EC2 GPU instance g2.2xlarge which has pre-installed NVIDIA GPU GK104 plus 22EC2 compute units (8 cores). Model was built in Keras library, backend as TensorFlow at beginning then on Theano.

Important features:

1. Training time on GPU with 25 epoch was more than 18 hours.
2. Model was trained with learning rate lr=1e-4 and drop-out rate as 0.5 to avoid over fitting.
3. As this problem relates to regression, accuracy was measured with MSE ie: mean square error between predicted angle and actual angle.
4. I also tried with preprocessing image data input with image augmentation and zca whitening, horizontal flip, vertical flip, but that hasn’t show any improvement. It might have created unwanted artifacts due to the texture of the road and limitation of the window size.
5. Another training technique from the NVIDIA paper was changing steer wheel angle in radian helps to improve performance.



**Evaluate performance:**

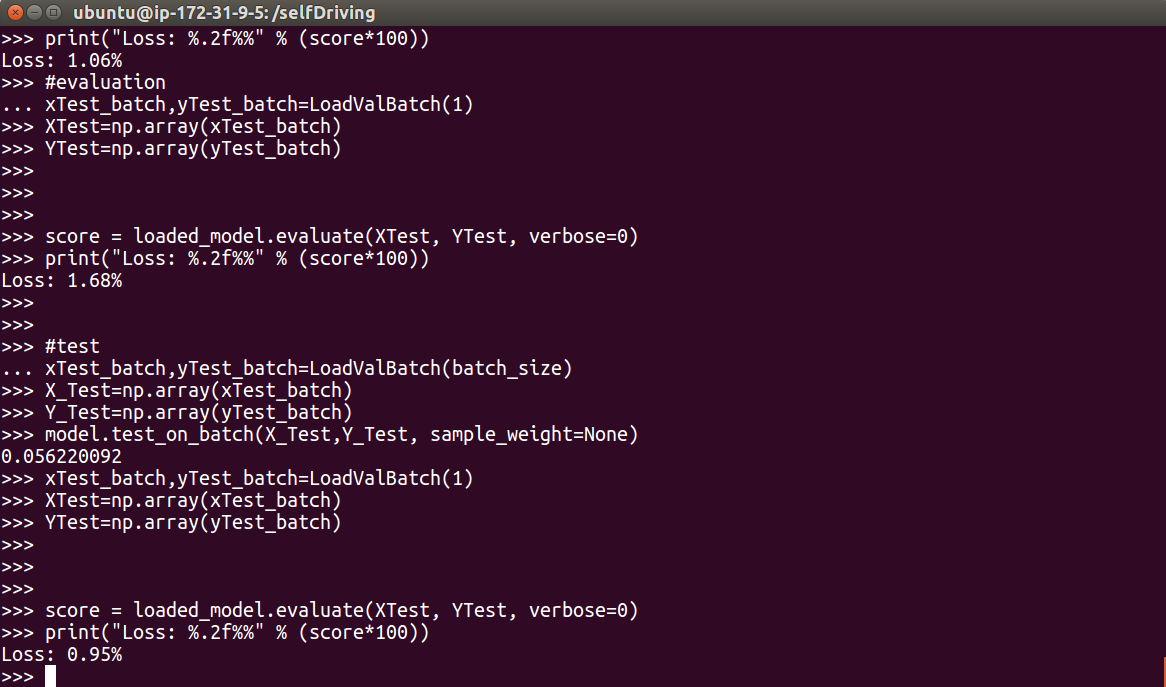


Evaluation score for batch validation: 0.0209681

Loss on steer wheel angle for single image:

error = 0.0168\*100= 1.68%

Accuracy = 98.32%



**Further Work:**

Although the results fared well for this project, further work would need to be done to get a model to perform on the road. Model needs to be trained on multiple GPU and with more than 32GM RAM and higher disk space and cache. Given more time, training two-stream models with a stack of optical flow images would have performed better.

**References:**

<https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf>

<http://www.nvidia.com/object/drive-px.html>

<https://arxiv.org/abs/1412.6980v8>

<https://keras.io/preprocessing/image/>

<http://sentiment-mining.blogspot.com/2015/08/the-difference-of-activation-function.html>

<https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html>

<https://medium.com/udacity/challenge-2-using-deep-learning-to-predict-steering-angles-f42004a36ff3#.2x614nbkt>

<https://aws.amazon.com/marketplace/pp/B01M0AXXQB#product-description>

<http://machinelearningmastery.com/check-point-deep-learning-models-keras/>

<http://machinelearningmastery.com/save-load-keras-deep-learning-models/>

<https://github.com/fchollet/keras/issues/68>