Convolutional Neural Network for End-to-End Driving

Sarvagya Pant
MS in CS
University of Alabama in Huntsville
Huntsville, Alabama
sp0090@uah.edu

Bidhya Nandan Sharma
MS in CS
University of Alabama in Huntsville
Huntsville, Alabama
bns0028@uah.edu

***Abstract*— Convolutional Neural Network (CNN) has proven to be of key importance in the field of image processing and autonomous driving. We trained a CNN to predict the steering angle for car based on the left, center and right images from the camera. With a minimal training, our model was sufficed to predict correct steering angle, enabling the simulated car run properly on the track. We used the Amazon WebServer (AWS) to train CNN model, and simulated car to test the performance of model. Our model automatically learns the important features based on images without having to explicitly write them.**

***Keywords* — Machine Learning, Neural Networks, Convolutional Neural Network, Self-Driving Cars.**

I. Introduction

Convolutional Neural Networks (CNN)[1][2] have made revolution in the field of image processing. Convolutional Neural Networks have proven to be very effective in the areas such as image recognition and classification. ConvNets are successful in identifying faces, objects and traffic signs apart from powering robots and self-driving cars [3]. The breakthrough feature of CNN is that features are learned automatically from training examples. In image processing, CNN perform very well because the convolution operation captures the 2D nature of images.

The rise in adoption of CNN has been increasing because of two recent developments [4]. First, large labeled datasets such as Large Scale Visual Recognition Challenge (ILSVRC) [5] have become available for training and validation. Secondly, CNN algorithms are implemented in the Graphics Processing Unit (GPU), which massively accelerates the learning process.

The background work for autonomous driving was done over 10 years ago at DARPA. In March 13 2004, first competition of DARPA was held at Mojave Desert. At the time, none of the robot vehicle completed the challenge. Later on, a Stanford racing team led by Sebastian Thrun made autonomous vehicle, Stanley, which utilized data from LIDAR, video cameras and GPS systems to find the correct position on terrain in order to move correctly. We present a CNN model that doesn't need to use GPS and LIDAR data. Instead, we will be using only the images captured to train and test our model [14].

II. Overview of CNN

LeNet by Yann LeCun was one of the very first convolutional neural network known [6].

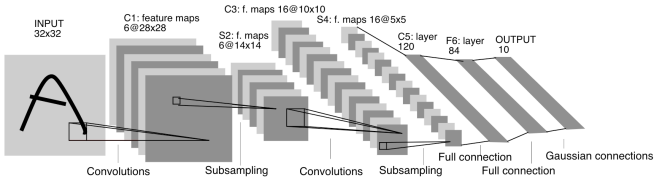


Figure 1 LeNet5

Figure 1 shows LeNet-5 Architecture, suggestive of four main operations:

1. Convolution
2. Non-Linearity (ReLu)
3. Pooling or Sub sampling
4. Classification (Fully Connected Layer)

A. Convolution

Convolutional layer applies convolution to the input and passes the output to the next layer. In this step, features are extracted from the image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data. This layer emulates the response of an individual neuron to visual stimuli [7].

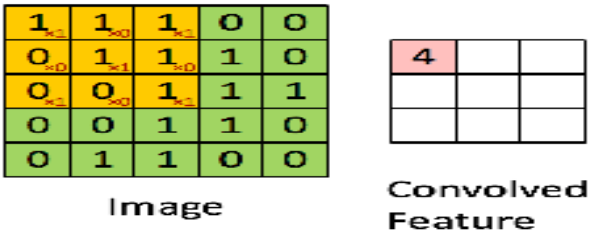


Figure 2 Convolution Operations

In Figure 2, green areas represent the pixels of image whereas yellow area is the 3X3 kernel, which is being convolved around the image. The result of convolution is 4. This matrix is known as Convolved Map or Feature Map or Activation

Map. In practice, a CNN learns the value of filters on its own during the training phase. However, we need to specify the number of filters, filter size, architecture of network, etc. before we begin the training phase. The size of the Feature Map is controlled by three parameters that needs to be decided before the convolution step [8]:

1) Depth: Depth corresponds to the number of filters we use for the convolution operation.

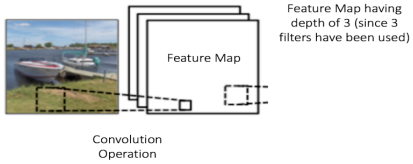


Figure 3 Feature Map Depths

Figure 3 is indicative of the convolution of original image using three distinct filters, generating three distinct feature maps, where the depth of feature map was three.

2) *Stride:* Stride is the number of pixel by which we slide our filter matrix over the input matrix. When the value is 1, we move one pixels at a time whereas for stride 2, the filter jumps 2 pixel at a time, sliding them around the image. A large stride would produce smaller feature maps.

3) Zero Padding: Zero padding might be important to maintain the uniformity in matrix, so that size of the feature map is produced accurately as required.

B. Non-Linearity (ReLu)

After the convolutional layer, we apply an additional operation known as Rectified Linear Unit (ReLu). It's given with equation:

$$f(x) = max(0, x). \quad (1)$$

We could use other functions such as *tanh* and *sigmoid.* ReLu is preferred to other functions because it trains the neural network several times faster without a significant penalty to generalization accuracy [9].

C. Pooling or Sub Sampling:

It's common to insert a Pooling layer in-between successive Conv layers in CNN. Pooling reduces the dimensionality of feature maps but retain the important information. Pooling can be of types: Max, Sum, Average, etc.

For example, let us consider a window of size 2X2 and stride of 2.

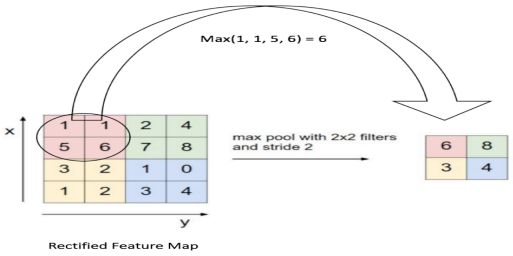


Figure 4 Pooling operation

In Figure 4, we can see the result of applying pooling on the data matrix. In case of max pooling, we will take maximum value of data seen in the window. Figure 4 will be generated as a result of:

max(1,1,5,6), max(2,4,7,8), max(3,2,1,2) and *max(1,0,3,4)*

By introducing pooling layers, input representations are made smaller, reducing the numbers of parameters and computation in the network and controlling the overfitting issues.

D. Classification (Fully Connected Layers):

After several convolutional and pooling layers, high-level reasoning in the neural network is done via fully connected layers. The term "Fully Connected" states that every neuron in previous layer is connected to every neuron in next layer.

III. Overview of End-to-end Driving Model

A. Data Collection and Preprocessing

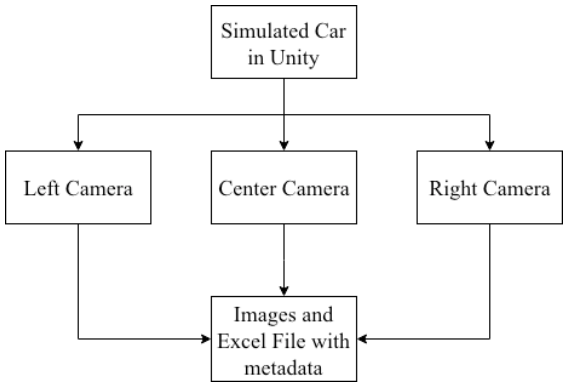


Figure 5 High-level view of Data Collection

Figure 5 shows the simplified block diagram of the collection systems of training data for End-to-End driving. In the simulated car, there are three cameras mounted for capturing left, center and right images. The training data consists of correct labeling of the image alongside the correct steering angle.

357	IMG/center_2016_12_01_13_33_14_230.jpg	IMG/left_2016_12_01_13_33_14_230.jpg	IMG/right_2016_12_01_13_33_14_230.jpg	0
358	IMG/center_2016_12_01_13_33_14_331.jpg	IMG/left_2016_12_01_13_33_14_331.jpg	IMG/right_2016_12_01_13_33_14_331.jpg	0.2722676
359	IMG/center_2016_12_01_13_33_14_433.jpg	IMG/left_2016_12_01_13_33_14_433.jpg	IMG/right_2016_12_01_13_33_14_433.jpg	0.3488158
360	IMG/center_2016_12_01_13_33_14_534.jpg	IMG/left_2016_12_01_13_33_14_534.jpg	IMG/right_2016_12_01_13_33_14_534.jpg	0.3488158
361	IMG/center_2016_12_01_13_33_14_636.jpg	IMG/left_2016_12_01_13_33_14_636.jpg	IMG/right_2016_12_01_13_33_14_636.jpg	0.0904655
362	IMG/center_2016_12_01_13_33_14_737.jpg	IMG/left_2016_12_01_13_33_14_737.jpg	IMG/right_2016_12_01_13_33_14_737.jpg	0

Figure 6 Samples of Training Data

Figure 6 displays the rows of data generated by the simulator engine. The first column is the name of image generated by center camera; second by the left camera and third by the right camera. In addition to the images, fourth column is the correct steering angle for those set of images. In addition to these images, augmentation of additional images could be important, done by flipping the image along Y-axis. In addition to image augmentation, there is a need for Image preprocessing. The image generated is of size 160X320.

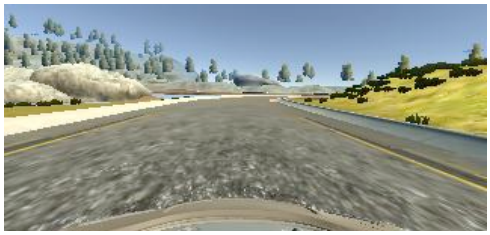


Figure 7 A sample center image

Figure 7 shows a center image obtained from the simulator engine. As we can see, there are a lot of noises in the image particularly the terrain and trees. In CNN, we don't want the network to recognize these noises as features, so the cropping of image to size 80X320 is necessary. Alongside cropping, random brightness was added to the image, enabling the model to learn the brightness of environment as well.

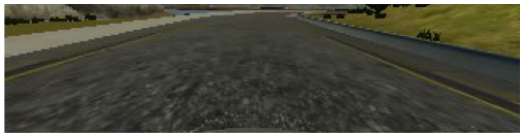


Figure 8 A cropped image with random brightness applied

Figure 8 is demonstrative of the cropped result and addition of random brightness to the image in figure 7. Consequently, unnecessary noises were eliminated, lowering the image sizes, which further helped in the decrement of training time.

B. Training and Testing Methodology

A block diagram of our training system is shown in Figure 9. Images from the left, center and right camera are first augmented and preprocessed. Then it is fed into the CNN, which computes a proposed steering angle. The value obtained is compared against the actual value to obtain error. This error is now backpropagated into the system, bringing CNN output closer to the actual output.

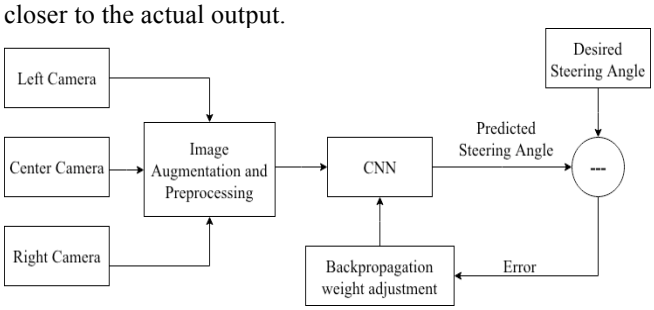


Figure 9 Training of the Neural Network

Figure 10 shows the testing of our model.

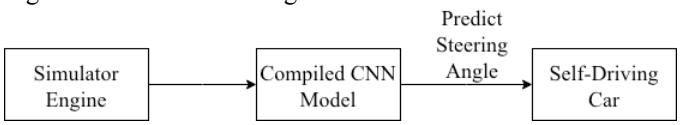


Figure 10 Testing of the CNN Model

In this figure, we can see that the simulator engine uses the compiled CNN model. By using our model, it predicts the steering angle per frame. This steering angle is used to run the self-driving car.

C. End-to-End Network Architecture

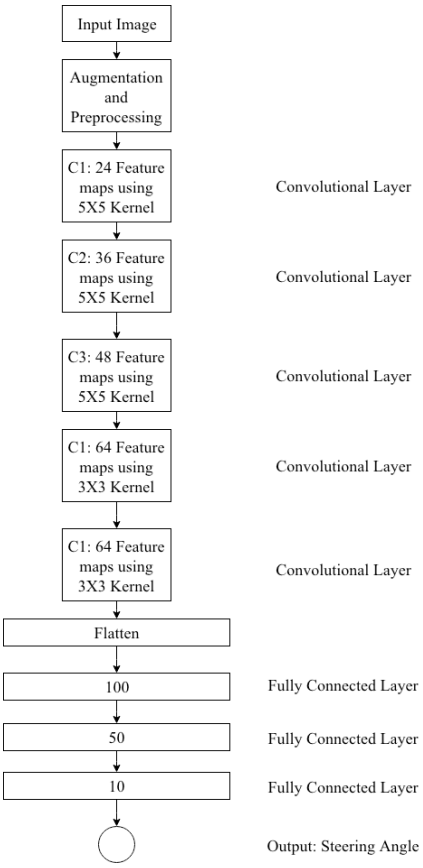


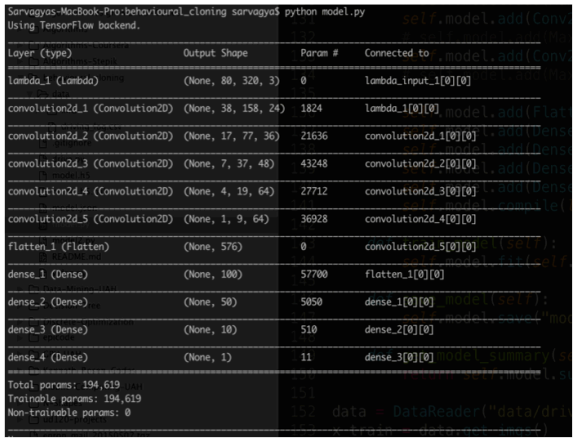
Figure 11 CNN Architecture for End-to-End Driving

We trained the weights in our network to minimize the mean squared error (mse) between the steering angle output by the network and actual data. In addition to it, we have used Adam Optimizer [10] during the compilation of the model. Figure 11 shows the architecture of our Convolutional Neural Network. The network consists of 8 layers, which comprises 5 convolutional layers and 3 fully connected layer. When an image is fed into the network, we augment the dataset by flipping it along Y-axis. In addition to it, we crop the original image of size 160X320 to 80X320 in order to reduce the noises like shape of tree, ponds, etc. Moreover, a random brightness is added to the image so that our network will also learn the lighting conditions of the track. Figure 7 shows the original image that consists of noises like terrains and trees. Furthermore, Figure 8 shows the image in which cropping and random brightness was applied.

The first three convolutional layers have stride of 2X2, kernel of size 5X5 and ReLu activation function. The Feature Maps produced by corresponding layers are 24, 36 and 48. Following these three layers, there are two convolutional layers that have stride of 2X2, kernel of size 3X3 and ReLu

activation function. Each of these layers was able to produce 64 Feature Maps.

We follow the five convolutional layers with three fully connected layers that predict the output steering angle. The number of neurons in each layer consists of 100, 50 and 10.



```
Sarvagyas-MacBook-Pro:behavioural_cloning sarvagya$ python model.py
Using TensorFlow backend.
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
lambda_1 (Lambda)                (None, 80, 320, 3)    0           lambda_input_1[0][0]
____________________________________________________________________________________________________
convolution2d_1 (Convolution2D)  (None, 38, 158, 24)   1824        lambda_1[0][0]
____________________________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 17, 77, 36)    21636       convolution2d_1[0][0]
____________________________________________________________________________________________________
convolution2d_3 (Convolution2D)  (None, 7, 37, 48)     43248       convolution2d_2[0][0]
____________________________________________________________________________________________________
convolution2d_4 (Convolution2D)  (None, 4, 19, 64)     27712       convolution2d_3[0][0]
____________________________________________________________________________________________________
convolution2d_5 (Convolution2D)  (None, 1, 9, 64)      36928       convolution2d_4[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 576)           0           convolution2d_5[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 100)           57700       flatten_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 50)            5050        dense_1[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                  (None, 10)            510         dense_2[0][0]
____________________________________________________________________________________________________
dense_4 (Dense)                  (None, 1)             11          dense_3[0][0]
====================================================================================================
Total params: 194,619
Trainable params: 194,619
Non-trainable params: 0
____________________________________________________________________________________________________
```

Figure 12 Parameters Learned by the model

Figure 12 shows the number of parameters learned by our architecture. The model will learn a total of 194,619 parameters, which is a good number of parameters to correctly predict the steering angle.

D. Implementation of Architecture

In order to implement the CNN architecture, libraries like Keras [11], Tensorflow [12] and OpenCV was chosen, using Amazon WebServer (AWS) G2.2XLarge to train the model on GPU. It had the following specifications needed to train the model:

- 15 GB NVIDIA GRID K520 GPU
- 26 compute Units
- 60 GB SSD

Keras, along with Tensorflow backend, can help in creating the CNN. We will illustrate some of the portions of our work during the Model building.

1) Image Augmentation and Preprocessing:

Consider the following piece of code that adds a random brightness in the image:

```
hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
rand = random.uniform(0.3,1.0)
hsv[:,:,2] = rand*hsv[:,:,2]
new_img = cv2.cvtColor(hsv, cv2.COLOR_HSV2RGB)
```

In this code, we can see that we are converting the RGB image first into the HSV (Hue, Saturation and Value) image. Then a uniform random number is chosen in the range of 0.3 to 1.0 and then the value is multiplied to the HSV image. Now we convert the HSV Image back to the RGB image.

2) CNN Model Creation

In addition to the preprocessing, let us consider one of the ways to create a convolutional layer. We present the code to add a convolution layer to the CNN model:

```
model = Sequential()
model.add(Lambda(lambda x: (x / 255.0) - 0.5,
input_shape=(80,320,3)))
model.add(Conv2D(24,5,5, activation="relu",
border_mode='valid', subsample =(2,2), W_regularizer =
l2(0.001)))
```

In the code above, we are creating a Sequential Model [13]. Now in order to apply normalization, we need to divide each pixel by 255 and subtract 0.5 from the resultant value. The code in the next line shows adding a Convolutional Layer in the Network. We are adding the first convolutional layer to produce 24 Feature maps with 5X5 Kernel size and a stride of 2X2, implementing ReLu activation function and L2 regularizer. Likewise, we could add other layers to the network.

3) Model Compile and Test

After the model is built, we need to compile and build it. This can be done using the functions provided in Keras known as *compile* and *fit*. In order to test the model inside the Unity simulator, we first need to save it. This can be done using the function known as *save.*

The Unity simulator can use a saved model by issuing the following python command:

python drive.py <saved-model>

Figure 13 Running instance of simulator

This commands opens a WSGI interface for the simulator. Through this interface, simulator can use the compiled model to predict the steering angle for the current frame of image.

Figure 13 shows the actual output of the program, where the model will predict the steering angle for the current frame, utilized by simulator to move the car by itself.

IV. Evaluation

Evaluation of our work is done using the simulation engine. We would like to present our accuracy, CPU vs. GPU performance and simulation in a track never seen before.

A. Accuracy

In figure 14, we can see the graph of validation loss vs. training loss. We can see that our training loss decreasing per epoch and after fifth epoch, decrease rate is almost constant.

However, Validation loss after the fifth epoch is increasing. This is because the model is overfitting instead of generalizing the problem. Hence, it was necessary for us to use the concept of Early Stopping. We trained our model until the fifth epoch. This model was then used for testing against the simulator.

Figure 14 Training vs. Validation Losses

B. CPU vs. GPU Performance

Figure 15 shows the performance of compiling our model in CPU and GPU. The training of model was done both on CPU and GPU. The CPU we used was of following specifications:

- 2.2 GHz Core i7 Processor
- Intel Iris Pro 1536 MB Graphics
- 16 GB RAM

We used Amazon Webserver (AWS) GPU with following specifications:

- 15 GB NVIDIA GRID K520 GPU
- 26 compute Units
- 60 GB SSD

As we can see in figure 15, training the model in GPU was at least six times faster than that of CPU. Using GPU, we have reduced the training time even for a medium size Convolutional Neural Network. Evidently, the rise of GPU has tremendously boosted the performance of Neural Networks in Deep Learning.

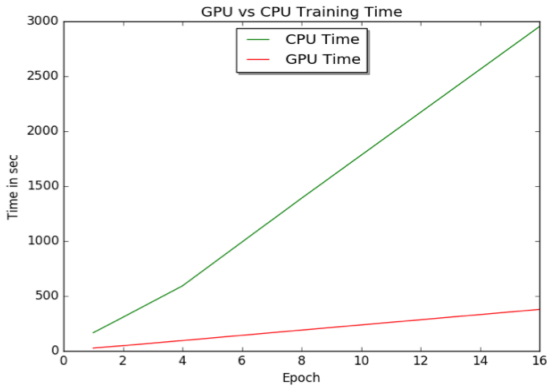


Figure 15 CPU vs. GPU Training Time

C. CPU vs GPU Precision

The capacity to perform floating-point arithmetic with high accuracy is a plus for optimization algorithms to achieve a better accuracy even after few numbers of training epochs. We have plots of training and validation loss in CPU and GPU. Figure 16, 17 shows that GPU (colored blue) is able to achieve lower loss than CPU (colored yellow) referring to the floating-point accuracy.

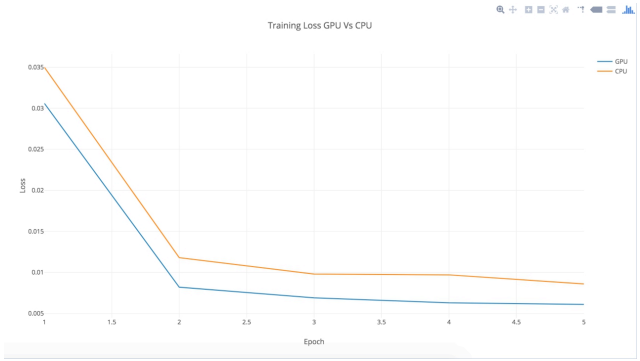


Figure 16 Training Loss in CPU vs. GPU

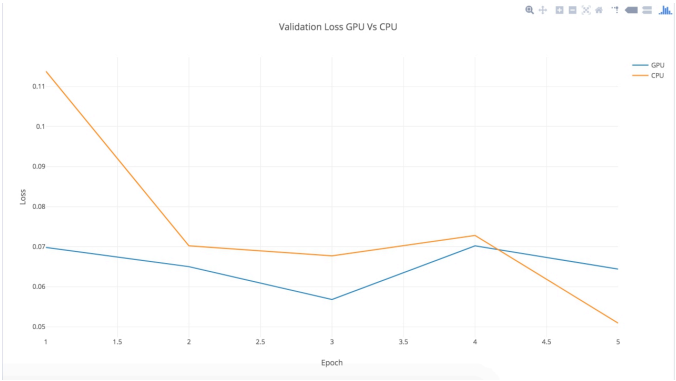


Figure 17 Validation Loss in CPU vs. GPU

D. Simulation in completely new track

We have tested our CNN model on a completely new track, which was not used in training step. Our model performed well on this track and was able to further generalize the problem. Here, figure 16, shows the snapshot of simulator engine on a completely new track.

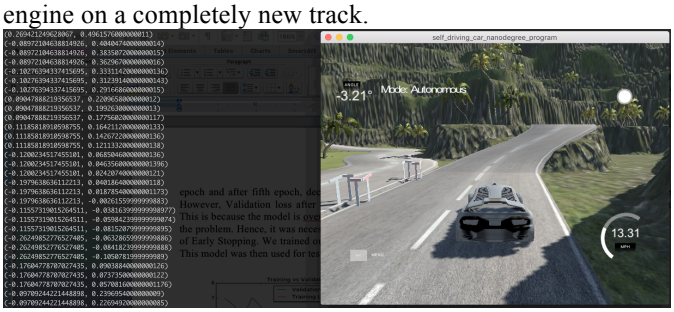


Figure 18 Test on completely new track

V. Future work

Currently the model only drives the car with the motive to keep it in track. As a part of future work, the model can be further improved to learn to stay in single lane. Furthermore, it can be also enhanced to predict throttle and brakes values, in addition to the steering angle being predicted. Currently, the car is able to drive in a road devoid of traffic, which is a potential area of enhancement. On that end, the project could be extended to incorporate learning traffic signs and symbols.

VI. Conclusion

Having successfully built and tested Keras CNN model to drive the Udacity's simulator in both GPU and CPU, there are something we would like to present. With high performance GPU, training a CNN model is at least six times faster than CPU even for a medium sized data. This performance will be evident when the data size grows very larger. The AWS GPU that we used also seemed to have high floating-point accuracy. Thus, we were able to achieve lower training-loss per epoch.

VII. PEER FEEDBACK

We received numerous of feedbacks from our peers. Some of the ideas given by the peers was very productive and were implemented in our project. For instance, a feedback was provided to use simple graphics, for which we chose the Unity Simulator provided by Udacity. Similarly, the other feedback was to present the outcomes of the project. For which we have demonstrated that the CNN model is able to drive the simulated car in both seen and completely new tracks. Additionally, this project was able to address the ongoing research in Computer Science known as "Deep Learning", which is more of a real-life GPU implementation rather than lower-level abstraction, based on an important feedback given by some of our peers.

Acknowledgment

This paper is the result of guidance and feedbacks that we received from Dr. Buren E Wells and other anonymous project reviewers. Besides this, we would also like to thank the creators and contributors of Keras, Tensorflow, and Udacity simulator.

References

[1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel.Backpropagation applied to handwritten zip code recognition. Neural Computation, Winter1989.

[2] *Convolutional Neural Netoworks*, en.wikipedia.org/wiki/Convolutional_neural_network.

[3] Karn, U. (2016, August 11). An Intuitive Explanation of Convolutional Neural Networks. Retrieved August 04, 2017, from https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[4] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., ... & Zhang, X. (2016). End to end learning for self-driving cars.

[5] Net-Scale Technologies, Inc. Autonomous off-road vehicle control using end-to-end learning, July 2004.Final technical report.

[6] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[7] "Convolutional Neural Networks (LeNet) – DeepLearning 0.1 documentation". DeepLearning 0.1. LISA Lab. Retrieved 31 August 2013. M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[8] CS231n Convolutional Neural Networks for Visual Recognition. (n.d.). Retrieved August 05, 2017, from http://cs231n.github.io/convolutional-networks/

[9] Krizhevsky, A.; Sutskever, I.; Hinton, G. E. (2012). "Imagenet classification with deep convolutional neural networks" (PDF). Advances in Neural Information Processing Systems. 1: 1097–1105.

[10] Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization.

[11] Chollet, F. (2015). Keras.

[12] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Ghemawat, S. (2016). "Tensorflow: Large-scale machine learning on heterogeneous distributed systems".

[13] Chollet, F. (n.d.). The Sequential model API. Retrieved August 06, 2017, from https://faroit.github.io/keras-docs/1.2.0/models/sequential/

[14] Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., ... & Lau, K. (2006). Stanley: The robot that won the DARPA Grand Challenge. Journal of field Robotics, 23(9), 661-692.