Prediction of Steering Angles for Self-Driving Cars Using Deep Learning

by Kevin Alejandro Dubón Aranda
Machine Learning Engineer Nanodegree - Udacity

Abstract

Self-Driving Cars will optimize our transportation infrastructure and will ultimately upgrade our lifestyles. Recent advancements in Machine Learning have allowed us to bring this technology closer to reality. The goal of this project was to utilize the power of Deep Machine Learning algorithms in order to develop a model that could predict steering angles almost exactly as a human being would control a car´s steering wheel. This was accomplished by training a Deep Neural Network composed of 9 layers. The model was trained for a relatively short term of 25 epochs, with a small dataset of less than 30 minutes of driving data, which allowed it to score a Root Mean Squared Error of 0.0558 on the testing dataset, surpassing the benchmark of 0.2068. In the end, this score demonstrated the ability of Deep Learning models to perform on a human-like manner with only a small training.

1. Definition

1.1 Project Overview

The development of autonomous vehicles has been one of the most important technological challenges of this decade. Self-driving cars will have a great impact on our society by eliminating car accidents, reducing traffic, optimizing fuel utilization, and also lowering the cost of the vehicles themselves. Self-driving car technology can also save unproductive time commuting to work, reduce car ownership (switching to an “On-Demand” usage), reduce parking lots needed (because cars won’t need to park), reduce mandatory car insurance (because cars will avoid accidents).

In recent years, there have been several major breakthroughs in the field of autonomous vehicles, all thanks to the implementation of “Deep” Machine Learning algorithms and techniques, such as Convolutional Neural Networks (CNNs).

CNNs [1] have revolutionized pattern recognition [2]. Prior to the widespread adoption of CNNs, most pattern recognition tasks were performed using an initial stage of hand-crafted feature extraction followed by a classifier. The breakthrough of CNNs is that features are learned automatically from training examples. The CNN approach is especially powerful in image recognition tasks because the convolution operation captures the 2D nature of images. Also, by using the convolution kernels to scan an entire image, relatively few parameters need to be learned compared to the total number of operations.

While CNNs with learned features have been in commercial use for over twenty years [3], their adoption has exploded in the last few years because of two recent developments. First, large, labeled datasets such as the Large Scale Visual Recognition Challenge (ILSVRC) [4] have become available for training and validation. Second, CNN learning algorithms have been implemented on the massively parallel graphics processing units (GPUs) which tremendously accelerate learning and inference.

This project demonstrates the ability to use the power of CNNs to control a car's steering only by processing image frames captured by an on-board camera.

1.2 Problem Statement

Current methods for autonomous steering require explicit decomposition of the problem, such as lane marking detection, path planning, and control, as shown in Figure 1. The primary motivation for this work is to avoid the need to recognize specific human-designated features, such as lane markings, guard rails, or other cars, and to avoid having to create a collection of "if, then, else" rules, based on observation of these features.

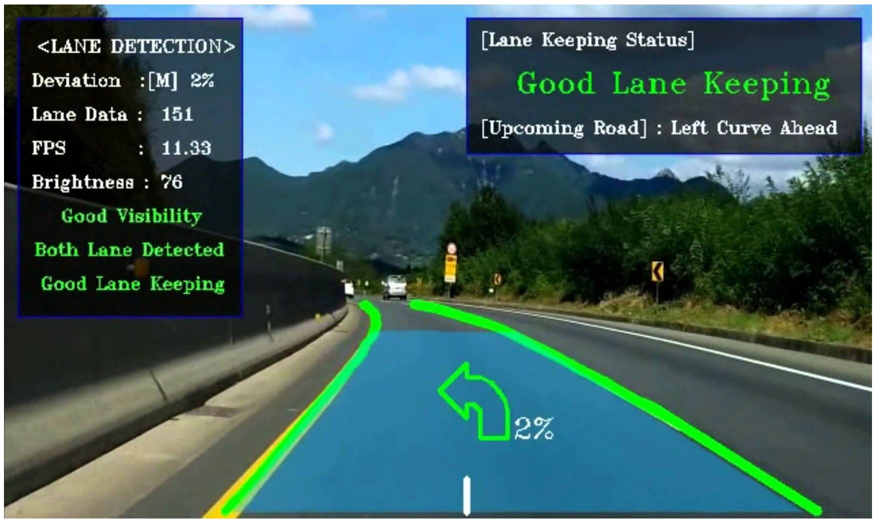


Figure 1. Example of lane marking detection, path planning, and control.

The basic goal is to use data collected from Udacity's Self Driving Car to build a model that can predict the steering angles for the vehicle with a relatively small training (~30 minutes of driving data). This problem is well suited for Convolutional Neural Networks (CNNs) that take in the forward facing images from the vehicle and output a steering angle. Since the predictions will be continuous data, this challenge can be treated as a regression problem.

End-to-end solutions like this, where a single network takes raw input (camera imagery) and produces a direct steering command, are considered the holy-grail of current autonomous vehicle technology, and are speculated to populate the first wave of self-driving cars on roads and highways. 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.

1.3 Metrics

In order to evaluate the effectiveness of our model's predictions, we must measure the output with some type of metric - in this case - RMSE. The root-mean-square deviation (RMSD) or root-mean-square error (RMSE) is a frequently used measure of the differences between values (sample and population values) predicted by a model or an estimator and the values actually observed. The RMSE [5] represents the sample standard deviation of the differences between predicted values and observed values and it is calculated as shown in Figure 2, where Ypred represents the predicted steering angle, Yref represents the observed steering angle, and N represents the total amount of predicted angles.

$$\mathrm{RMSE} = \sqrt{\sum \frac{(y_{pred} - y_{ref})^2}{N}}$$

Figure 2. Formula to calculate RMSE.

2. Analysis

2.1 Data Exploration

Training for this model was done using Udacity's Challenge #2 Dataset, which is based on driving data from San Mateo, CA to Half Moon Bay, CA (curvy and highway driving). This dataset contains image frames captured by a set of on-board cameras, where each image is associated with a certain steering angle and timestamp. Stationary segments and lane changes in the dataset have been removed by Udacity.

All sensor data including imagery and steering wheel angles is provided in the ROSbag format. Images were extracted into PNGs via rwightman's udacity-driving-reader [6], which also provided a CSV of interpolated steering wheel angles for the provided timestamps/frame IDs. Even though the imagery is from three different cameras (left, center, right), only the center images were used.

The Challenge #2 Dataset consists of 6 parts, as shown in Table 1. Parts 1, 2, 5 and 6 were used for training, while Part 4 was used for validation, and Part 3 for testing (as required by Udacity's Challenge #2). Samples of this dataset are shown in Figure 3 and Table 2, which show images as well as their corresponding timestamps and steering angles. Figure 4 helps us view the quality of the data by showing a quick overview of the variation of the steering angles across the dataset. The segments that appear as straight horizontal lines represent straight or constant driving, and the ups and downs represent curvy driving(which is good for training a steering model).

Dataset Part	Length (seconds)	Description
1	221	Direct sunlight, many lighting changes.Good turns in beginning, discontinuous shoulder lines, ends in lane merge, divided highway.
2	791	Two lane road, shadows are prevalent, traffic signal (green), very tight turns where center camera can't see much of the road, direct sunlight, fast elevation changes leading to steep gains/losses over summit. Turns into divided highway around 350s, quickly returns to 2 lanes.
3	281	Two lane road in sunlight and shadows. Ends when divided highway begins.
4	99	Divided highway segment of return trip over the summit.
5	212	Guardrail and two lane road, shadows in beginning may make training difficult, mostly normalizes towards the end.
6	371	Divided multi-lane highway with a fair amount of traffic.

Table 1. Description of Udacity's Challenge #2 Dataset.

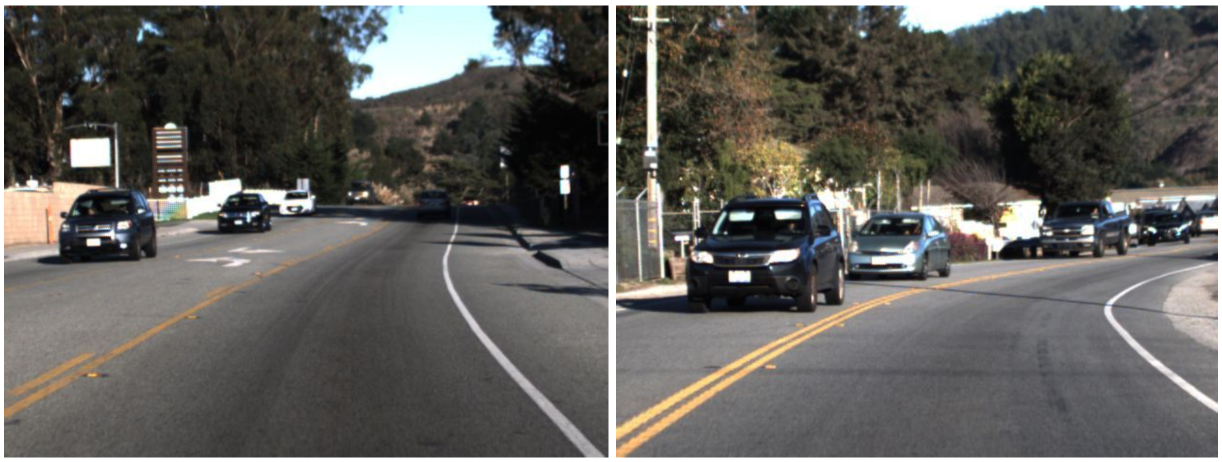
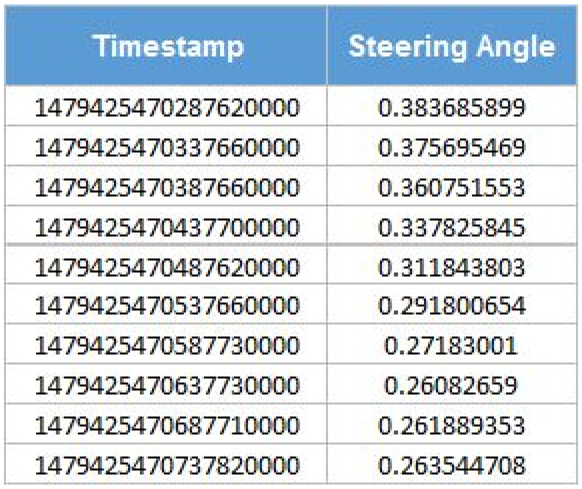


Figure 3. Image samples from Udacity's Challenge #2 Dataset.



Timestamp	Steering Angle
1479425470287620000	0.383685899
1479425470337660000	0.375695469
1479425470387660000	0.360751553
1479425470437700000	0.337825845
1479425470487620000	0.311843803
1479425470537660000	0.291800654
1479425470587730000	0.27183001
1479425470637730000	0.26082659
1479425470687710000	0.261889353
1479425470737820000	0.263544708

Table 2. Samples of image timestamps and steering angles from Udacity's Challenge #2 Dataset.

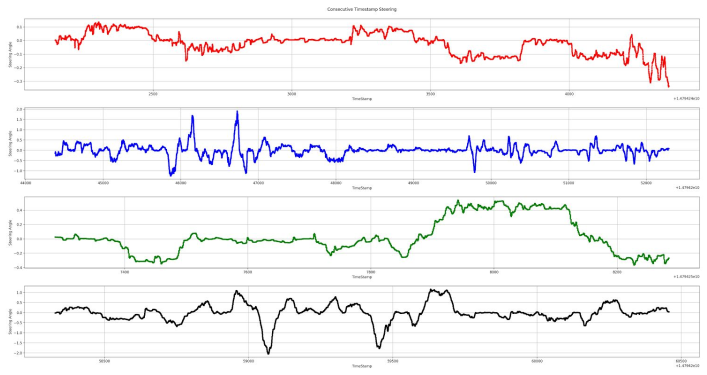


Figure 4. Variation of steering angles across Udacity's Challenge #2 Dataset.

2.2 Algorithms and Techniques

The use of Neural Networks for machine learning is commonly known as Deep Learning, due to their ability to extract "deeper" information patterns from the given data by simulating how real neurons work.

Convolutional Neural Networks (CNNs)

CNNs are a category of Neural Networks that have proven very effective in areas such as image recognition and classification. CNNs have been successful in identifying faces, objects and traffic signs apart from powering vision in robots and self driving cars. CNNs are designed to perform feature extraction by reducing image dimensionality. Figure 5 shows an example of how a CNN architecture reduces the dimensionality of an input image through a series of convolutions, until it generates a single output at the end.

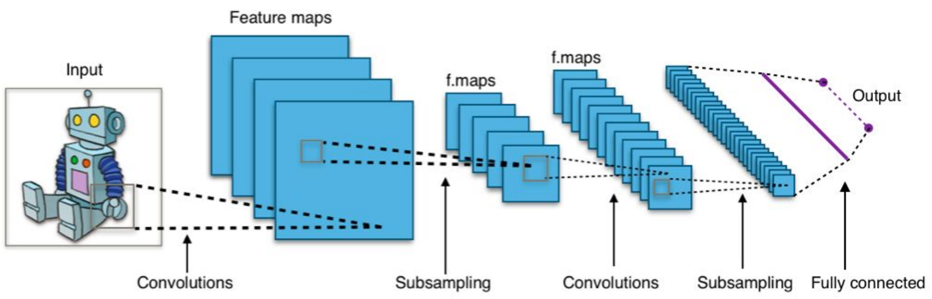


Figure 5. Dimensionality reduction through convolutional layers.

The way Convolutional Layers work is by filtering through small portions of an image. In order to do this, a convolutional layer requires 2 main parameters: Kernel and Stride. The kernel is the size (in pixels) that each convolution will filter through, while the stride refers to the number of pixels that we are shifting each time we move our filter. A stride of 1 makes the output roughly the same size as the input. A stride of 2 means it's about half the size. Figure 6 shows an example of a convolution filtering through a 7x7 image with a 3x3 kernel and strides of 1x1(no stride) and 2x2. It can be seen that the strided convolutions dramatically reduce the size of the image (more feature extraction), while the non strided ones remain about the same (less feature extraction).

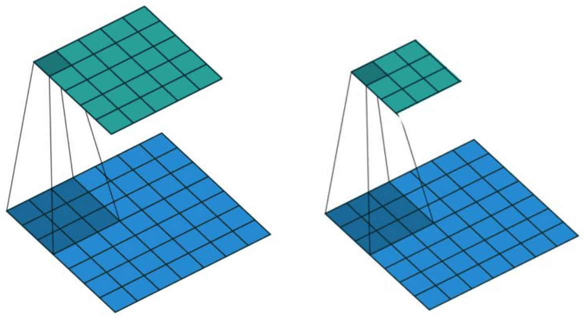


Figure 6. No Stride (left). Stride 2x2 (right)

Optimizers

One way in which CNNs differ from the standard layers is by sharing its weights across space. In this way, a convolutional layer glides repeatedly, or "convolutes" over an image, updating its weights with each convolution. These weights are then trained through backpropagation, which is a way of computing gradients of expressions through recursive application of the chain rule. In this way, we can use the loss function in order to quantify the quality of any particular set of weights W. The goal of optimization is to find W that minimizes the loss function. For this project, the "adam" optimizer was chosen, which is one of the standard optimizers from the Keras library and is described in the implementation section.

Dropout

Dropout is a simple and powerful regularization technique for neural networks and deep learning models. It works by randomly selecting neurons and ignoring them during training. They are “dropped-out” randomly, as shown in Figure 7. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

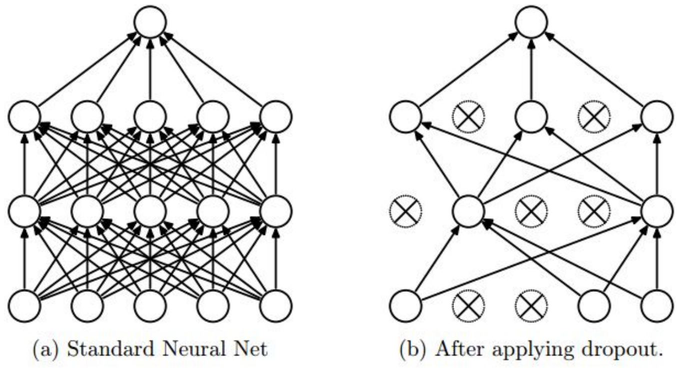


Figure 7. Dropout functionality.

Data Pre-processing

In much of machine learning, data used for training and inference undergoes a preprocessing step [7], where multiple inputs (such as images) are scaled to the same dimensions and stacked into batches. This lets high-performance deep learning libraries like TensorFlow[8] run the same computation graph across all the inputs in the batch in parallel.

Training

The weights of a neural network need to be trained in order to minimize the error between the input and output values (in this case, the steering command output by the network and the command of the human driver). Training of deep neural networks is done by epochs. Each epoch passes batches of pre-processed images through the model in order to train its weights, and provides values for training loss and validation loss. After training, the weights with the lower validation loss are passed on to the model in order to predict the output of the test dataset.

2.3 Benchmark

In order to ease the evaluation process, Udacity provided a benchmark so that participants can compare their model´s performance. For reference, an example file filled with '0' for every steering prediction scores a RMSE value of 0.20678229236453141. This means that an accurate prediction model should score a RMSE much lower than 0.2068.

3. Methodology

3.1 Data Preprocessing

Some additional pre-processing steps were applied to the input images in order to help our deep learning model extract information out of the images. The pre-processing steps applied are the following:

1. The raw images in the dataset are of size 640 x 480. In the final model, images were resized to 256 x 192 pixels.
2. Images were converted from RGB color format to grayscale because the RGB colors on the images do not provide relevant information to help predict steering angles.
3. Computed lag 1 differences between image frames and used 2 consecutive differenced images. For example, at time t I used [x_{t} - x_{t-1}, x_{t-1} - x_{t-2}] as input where x corresponds to the grayscale image. No future frames were used to predict the current steering angle.

All these steps help reduce the size of the data and also helps to magnify more relevant data, which in turn allows the model to run faster and more accurately. This data pre-processing was applied to every image in the training, validation, and test subsets. Figure 8 shows an example of an image after being pre-processed. As it can be seen, RGB colors were removed and only some edges can be seen. These edges help the model identify patterns such as street lanes and other cars.

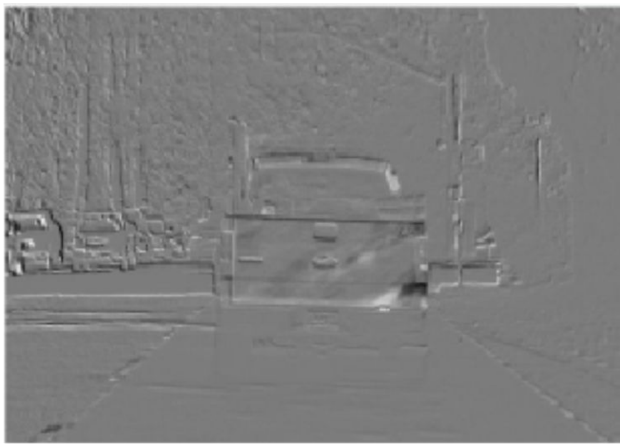


Figure 8. Sample of a pre-processed image.

3.2 Implementation

This project was developed with Keras[9] on top of Tensorflow, and its implementation can be found on GitHub. Data pre-processing, model training and testing were done using the the recent P2 EC2 instances from Amazon Web Services, which are specialized for Machine Learning computation.

Training

The weights of this network were trained in order to minimize the mean squared error between the steering command output by the network and the command of the human driver. This model was trained for series of various epochs, ranging from 10 to 50, with batches of 32, 64, and 100 images.

Model

The main structure of this model uses the power of deep learning and is based on NVIDIA's research paper titled "End to End Learning for Self-Driving Cars"[10]. The network's architecture is shown in Figure 9 and it consists of 9 layers, including 5 convolutional layers and 4 fully connected layers.

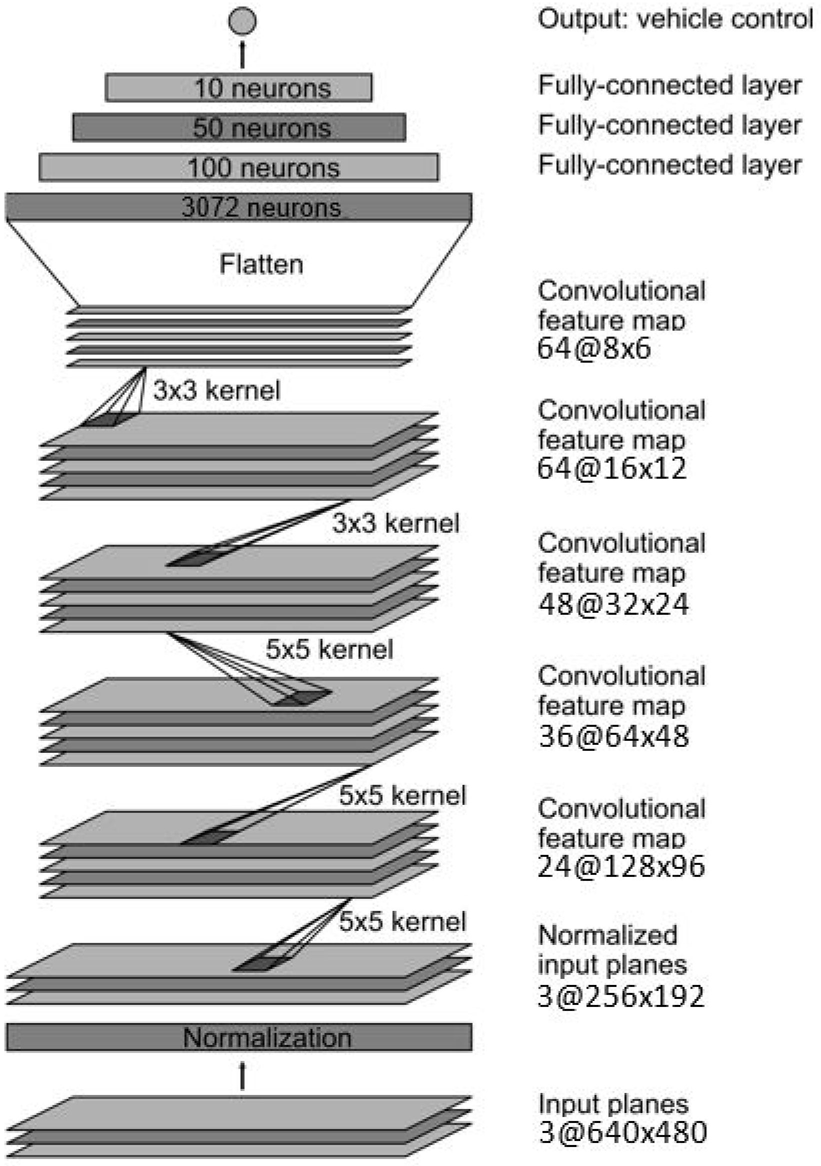


Figure 9. Main Deep Learning Architecture (Source: NVIDIA)

The convolutional layers were chosen empirically through a series of experiments that varied layer configurations. The network uses strided convolutions in the first three convolutional layers with a 2x2 stride and a 5x5 kernel and a non-strided convolution with a 3x3 kernel size in the last two convolutional layers.

The network's five convolutional layers are followed by three fully connected layers leading to an output control value which is the inverse turning radius. Although the fully connected layers are designed to function as a controller for steering, it's important to note that by training the system end-to-end, it is not possible to make a clean break between which parts of the network function primarily as feature extractors and which serve as the controller.

The optimizer used for this model is the "adam" optimizer from Keras, which consists of the following parameters:

- Learning Rate: 0.001
- Beta_1: 0.9
- Beta_2: 0.999
- Epsilon: 1e-08
- Decay: 0.0

3.3 Refinement

At the beginning, the model worked by taking in raw images as input, but it turned out to be very inaccurate with a RMSE greater than 0.20. Later on, image augmentation was implemented in order to pre-process all the input data, which allowed the model to perform better, with a RMSE lower than 0.10.

Several changes were applied to the model's architecture to try to improve it, but actually decreased its performance: 1. Addition of a 6th Convolution layer identical to the 5th; 2. Enlargement of the final FC layers from (100,50,10) to (256,128,64). Both of these changes caused the model to overfit more rapidly and yielded higher RMSEs. These 2 models were compared against the first version, training all 3 for a series of 10 and 25 epochs with batch size of 32, 64, and 100. In the end, the first version turned out to yield the best results when being trained for 25 epochs with batches of 32 images.

One of the most important things in Machine Learning models is the data they are fed with. In this case, we took a step further to clean the dataset, removing segments which remained constant for long periods. In the end, the training set was reduced from 33,800 to 31,800 image frames. This filtering allowed the model to overfit less rapidly, making more accurate predictions, and therefore allowed it to score a RMSE lower than 0.06.

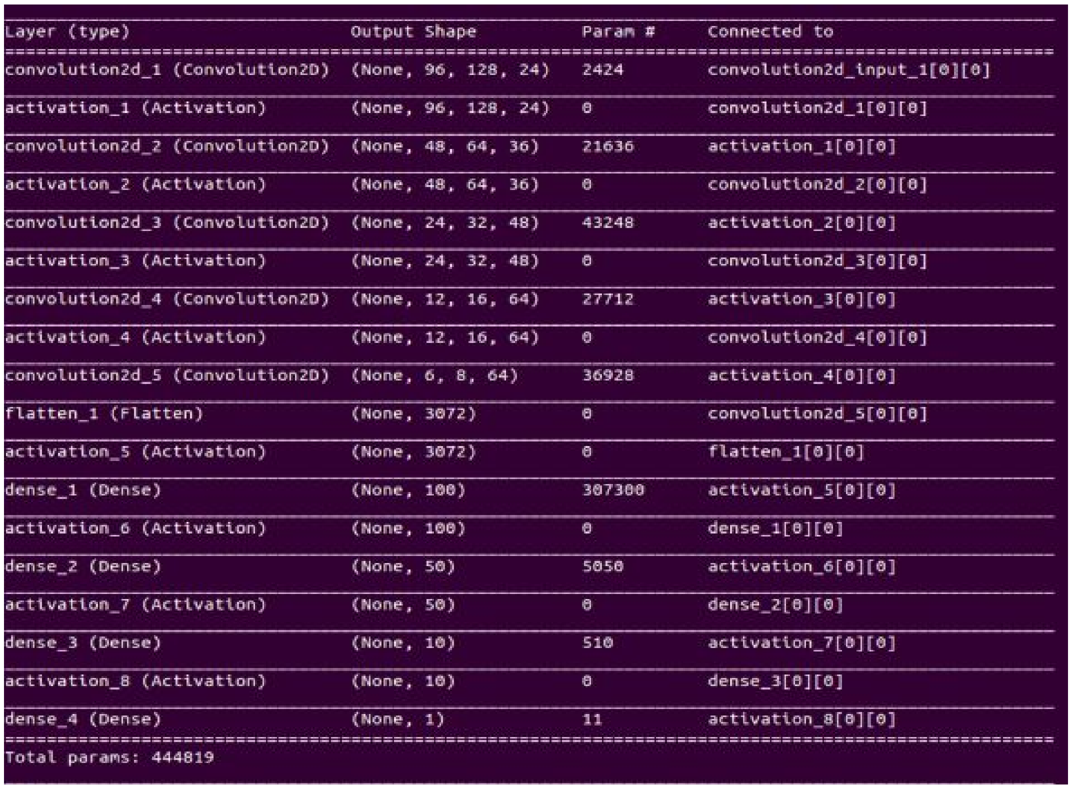
As a final improvement, dropout was added before the FC layers in order to help reduce overfitting. This addition allowed the model to get a validation loss of 0.00161 during training, leading to a final RMSE of 0.0558.

4. Results

4.1 Model Evaluation and Validation

As mentioned in Section 3.2, the final structure of this model consists on a combination of 9 layers, with a total of 444,819 parameters, as shown in Figure 10. What this means is that there are over 400 thousand parameters that can be tuned for every image that passes through the neural network in order to predict the steering angle. Some of these parameters might serve to identify curves, street lanes, cars, or other elements, but since this is an end-to-end solution, we are unable to visualize what goes on inside the layers.

This model was trained for series of 10, 25, and 50 epochs. The best results were obtained from a combination of 25 epochs with batches of 32 images, which provided the lowest validation loss of 0.00161 on the 19th epoch. With the current configuration, the model managed to predict steering angles with an RMSE of 0.0558.



```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
convolution2d_1 (Convolution2D)  (None, 96, 128, 24)   2424        convolution2d_input_1[0][0]
____________________________________________________________________________________________________
activation_1 (Activation)        (None, 96, 128, 24)   0           convolution2d_1[0][0]
____________________________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 48, 64, 36)    21636       activation_1[0][0]
____________________________________________________________________________________________________
activation_2 (Activation)        (None, 48, 64, 36)    0           convolution2d_2[0][0]
____________________________________________________________________________________________________
convolution2d_3 (Convolution2D)  (None, 24, 32, 48)    43248       activation_2[0][0]
____________________________________________________________________________________________________
activation_3 (Activation)        (None, 24, 32, 48)    0           convolution2d_3[0][0]
____________________________________________________________________________________________________
convolution2d_4 (Convolution2D)  (None, 12, 16, 64)    27712       activation_3[0][0]
____________________________________________________________________________________________________
activation_4 (Activation)        (None, 12, 16, 64)    0           convolution2d_4[0][0]
____________________________________________________________________________________________________
convolution2d_5 (Convolution2D)  (None, 6, 8, 64)      36928       activation_4[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 3072)          0           convolution2d_5[0][0]
____________________________________________________________________________________________________
activation_5 (Activation)        (None, 3072)          0           flatten_1[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 100)           307300      activation_5[0][0]
____________________________________________________________________________________________________
activation_6 (Activation)        (None, 100)           0           dense_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 50)            5050        activation_6[0][0]
____________________________________________________________________________________________________
activation_7 (Activation)        (None, 50)            0           dense_2[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                  (None, 10)            510         activation_7[0][0]
____________________________________________________________________________________________________
activation_8 (Activation)        (None, 10)            0           dense_3[0][0]
____________________________________________________________________________________________________
dense_4 (Dense)                  (None, 1)             11          activation_8[0][0]
====================================================================================================
Total params: 444819
____________________________________________________________________________________________________
```

Figure 10. Parameters of the Deep Learning Model. (Screenshot of Keras table)

4.2 Justification

With the current configuration, the model managed to predict steering angles with an **RMSE of 0.0558**. As mentioned in Section 2.4, the benchmark for this problem is an RMSE value of 0.2067. Although the model does not predict angles with perfection, an RMSE of 0.0558 means it performs really well, with a very small difference compared to human-controlled steering.

The predictions for the steering angles can be visualized in a simulator-like format in this video, as shown in Figure 11.

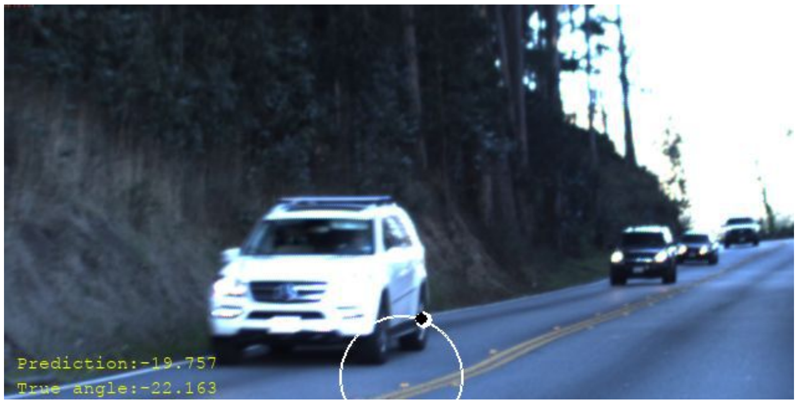
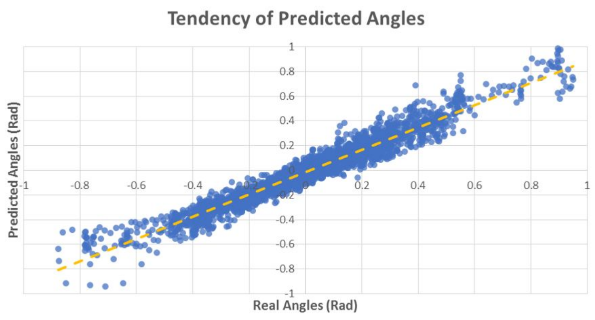


Figure 11. Final Visualization of Steering Model.

A deeper representation of the results is shown in Figures 12 and 13, which show the tendencies for both, the predictions made by our model and the errors from those predictions. Figure 12 shows a positive trend for the predictions, which means that as the real angles increase, so will the predicted angles, and vice versa. Figure 13 shows an almost-neutral tendency for the errors between the predictions and the real angles, which means that the error rate remains relatively the same whether the real angles increase or decrease.



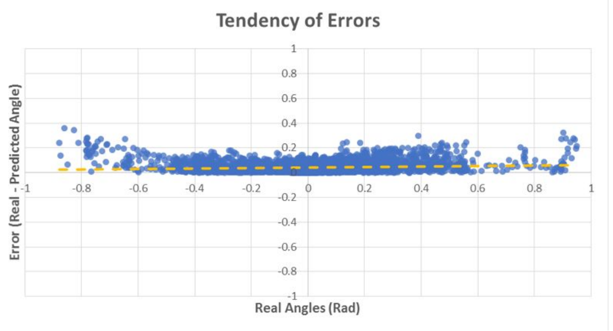


Figure 12. Tendency of Predicted Angles **Figure 13.** Tendency of Errors

5. Conclusion

This model demonstrates that CNNs are able to learn the entire task of lane and road following without manual decomposition into road or lane marking detection, semantic abstraction, path planning, and control. A small amount of training data from **less than 30 minutes of driving** was sufficient to train the model to operate on a human-like way for steering a car. The CNN is able to learn meaningful road features from a very sparse training signal (steering alone). As an example, the system learned to detect the outline of a road without the need of explicit labels during training. Figure 14 shows a comparison between the Real angles (blue) and the predicted angles (orange). It can be seen that the predictions manage to stick tightly to the real angles, with only a small variation on greater angles.

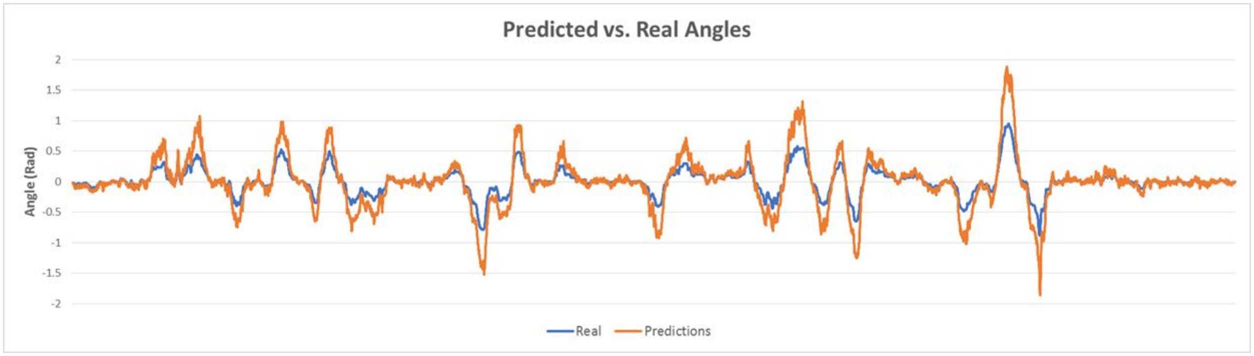


Figure 14. Final Comparison of Predicted Steering Angles.

Reflection

Besides being really challenging, this project has given me a taste of the power of deep learning to solve big, real-world problems which can benefit millions of people. I find it really fascinating to be able to control a vehicle's steering only by looking at image frames. Also, being able to use the power of AWS's Tesla K80 GPUs was a cheap way to get the juice out of ConvNets and also reduced training time dramatically.

Improvement

Even though this model works well, some improvements can be made to further optimize it.

1. More work is needed to improve the robustness of the network, to find methods to verify the robustness, and to improve visualization of the network-internal processing steps.
2. Training with different types of road conditions such as road type or weather could allow the model to learn how to drive on a wider variety of terrains.
3. Training on a wider range of curves -with a larger dataset- could allow the model to reduce the error for large steering angles.
4. Use transfer learning with different existing structures, such as alexnet to improve the model's quality.

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,* 1(4):541–551, Winter 1989. URL: http://yann.lecun.org/exdb/publis/pdf/lecun-89e.pdf.

2. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 25, pages 1097–1105. Curran Associates, Inc., 2012.
URL:http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

3. L. D. Jackel, D. Sharman, Stenard C. E., Strom B. I., , and D Zuckert. Optical character recognition for self-service banking. AT&T Technical Journal, 74(1):16–24, 1995.

4. Large scale visual recognition challenge (ILSVRC).
URL:http://www.image-net.org/challenges/LSVRC/.

5. Wikipedia.org RMSE metric.
URL:https://en.wikipedia.org/wiki/Root-mean-square_deviation

6. Udacity Driving Reader. Quick docker based reader for udacity rosbag self-driving dataset. URL:https://github.com/rwightman/udacity-driving-reader

7. TensorFlow Fold: Deep Learning With Dynamic Computation Graphs.
URL:https://research.googleblog.com/2017/02/announcing-tensorflow-fold-deep.html

8. Tensorflow: An open-source software library for Machine Intelligence.
URL:https://www.tensorflow.org/

9. Keras: Deep Learning library for Theano and TensorFlow. URL: https://keras.io/

10. Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, Karol Zieba. End to End Learning for Self-Driving Cars. URL:https://arxiv.org/abs/1604.07316