# Machine Learning Engineer Nanodegree

## Capstone Report

Ahmad Waly
October 10th, 2019-10-10

# Report

---

## I. Definition

### Domain Background

An autonomous car is a vehicle that can guide itself without human conduction. Imagining a car driving itself without any human interference was science-fiction and was first introduced in movies. Today, as we talk there are many companies out there building autonomous cars.

Experiments have been conducted on automated driving systems (ADS) since at least the 1920s;[8] trials began in the 1950s.

At 2016, Nvidia has proposed a new approach to build and end to end system that can drive a car using a cnn to map raw pixels to steering angles

### Problem Statement

This project focus on controlling the steering wheel of a fully autonomous vehicle The system should predict the steering angle according to the road image

Using this approach the system act End-to-End : it learn the internal representations of the processing steps without being explicitly trained to do this so the problem won't be explicitly decomposed to lane marking , path planning ,control … etc , the end to end system will supposedly optimizes all this and learn it

The objective of this project is that the system can drive itself the system learns to drive in traffic on local roads with or without lane markings and on highways. It also operates

in areas with unclear visual guidance, This will be viewed as a supervised learning regression problem.

## Evaluation Metrics

RMSE will be used as evaluation metric. RMSE is the popular choice for a metric for regression problems , it is the root of the average of squared errors , the formula is :

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

The approach in the project is to teach a CNN to calculate the Steering wheel angle This approach, is a regression-type problem, made sense to use mean-squared error to minimize loss, meaning the difference between the actual angles and the model's prediction (MSE uses the mean of all the squared differences to calculate loss). The final approach I used also utilized MSE, as I used convolutional neural network to predict the angle , Using MSE here meant minimizing the loss between the predicted angle and the True labelled angle which makes it a very good candidate

## Datasets and Inputs

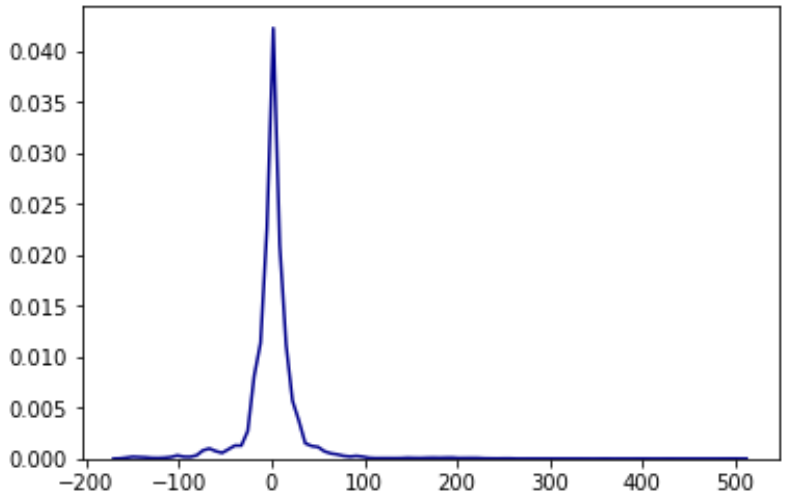
The dataset i chose is provided through this link https://github.com/SullyChen/driving-datasets i chose the second one it includes approximately 45000 image which has one label which is steering angle I will split it into train , validation and test

Dataset is a set of rgb images with dimension (256,455,3)


Fig (1)-Sample from dataset

## Exploratory Visualization

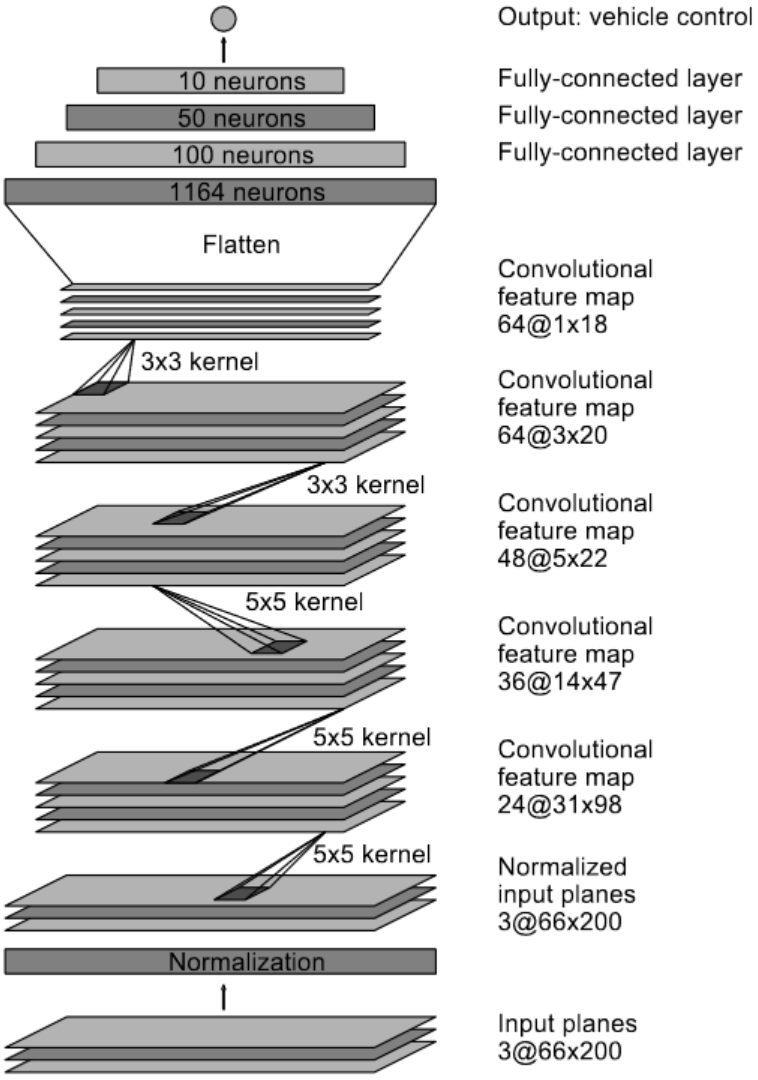
After getting insight about what the dataset represents here is a density plot that represents the range of angles in dataset it's obvious that angle in range[-30,30] contain much higher samples so I will try to increase the larger angles with augmentation technique




## Algorithms and Techniques

As mentioned in the paper and in the introduction here ; I will use deep learning and convolutional neural network, the convolutional part of the network supposedly learn features from image then these features are fed to the fully connected layers which act as the classifier part of the image

- The network archtitecture is as the following figure :




- The optimization algorithm used for learning process is ADAM which is a mix between Adaptive Gradient algorithm (AdaGrad) and root mean square propagation ( RMSPROP), The Adagrad maintains a per-parameter learning rate that improves performance on sparse gradients
- During training data is loaded in batches to avoid memory overload
- The model is implemented in keras
-

## Benchmark Model

I implemented a cnn that contains 2 conv layer with 2 filters each and trained it for 4 steps so that it's so naiive I chose this for the ease of implementation an testing

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_7 (Conv2D) | (None, 31, 98, 2) | 152 |
| conv2d_8 (Conv2D) | (None, 14, 47, 2) | 102 |
| flatten_4 (Flatten) | (None, 1316) | 0 |
| dense_7 (Dense) | (None, 15) | 19755 |
| dense_8 (Dense) | (None, 1) | 16 |

Total params: 20,025
Trainable params: 20,025
Non-trainable params: 0

It scored 6.35 Mean Squared error on testing set

# Methodology

## Data Preprocessing

To get the data ready there are few steps that have been done

### -Data Augmentation

**Flipping** : all images that has angle wider than 30 was flipped and added to dataset with new fipped label too


**Shuffling**

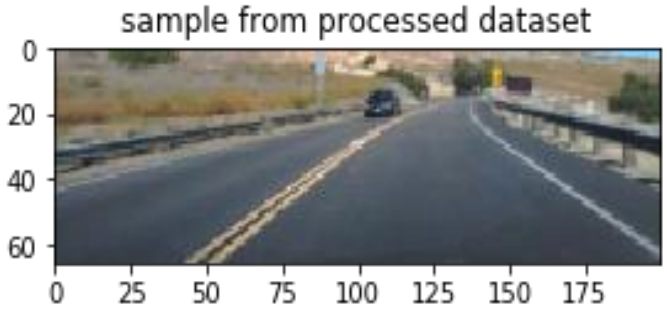
since data is collected in a sequence so to avoid the redundancy of any frame I shuffled the whole dataset

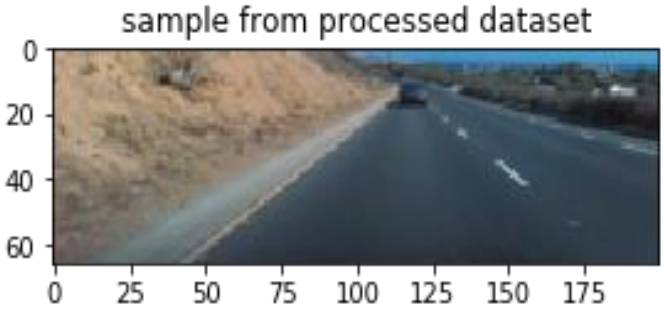
**Splitting**

I split the data into 3 chunks one for training , validation and testing with ratio 80:10:10 respectively

**Cropping and resizing**

While generating batches from data in training the upper region of the image is dropped so that we only have the region of interest in image is used then it is resized to be (66,200,3)



**Input normalization**

The raw image pixels are in range 0 -> 255 so as a normalization step before feeding images as input to the model to help the training to converge faster all pixels are normalized by dividing each pixel by 255

## Implementation and Model Training

The project was coded and executed locally and the libraries used on it are :

Keras with tensorflow backend , scipy and numpy for reading dataset and normalizing input , opencv for flipping , seaborn and matplot lib for visualizations

The architecture I used and trained from scratch is called PILOTNET it has almost 1.5 million parameters , I trained it for 8 epochs on almost 40000 image

For first 3 layers of convolutional layers kernel size was 5x5 with a stride 2 , while the last 2 were 3x3 with no strides

I used dropout as a regularization technique in fully connected layers to avoid over fitting

**Problems met**

At first when I was training the model stopped learning very early then I searched for the problem and found out that it's called a dead relu where all activations are zero so I changed it to elu instead

Also I tried default keras initialization but it wasn't the best choice so I searched and I found in the repo of sulychen provided in references that he used truncated normal initialization for weights

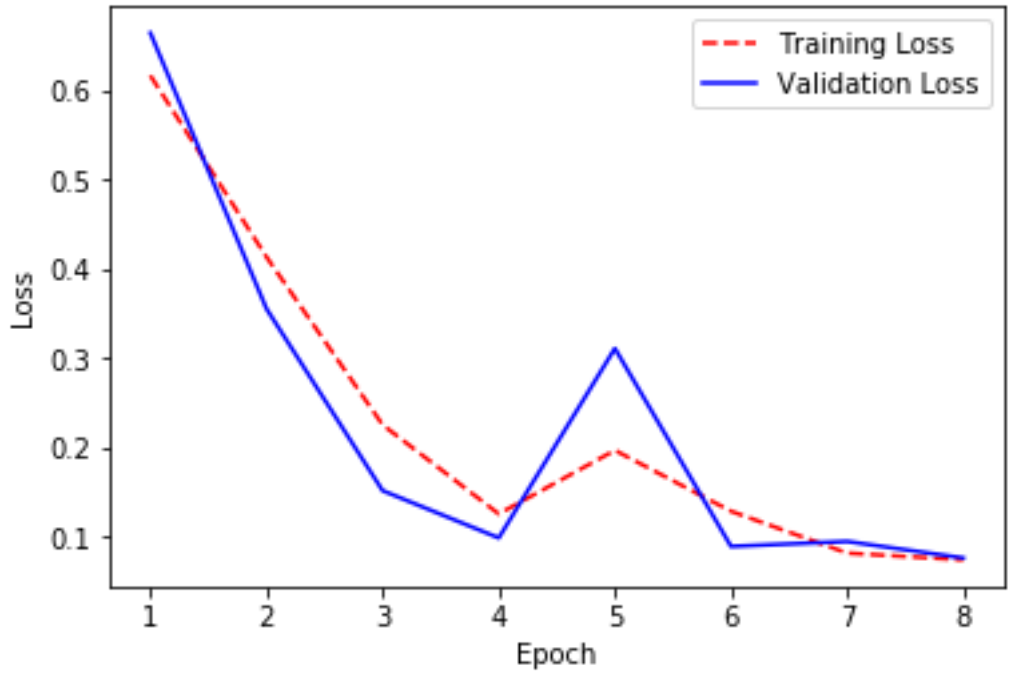
Also I tried different number of epochs and found out that 8 epochs are the best choice as after that it overfits on data

---

# Results

## Model evaluation and validation

At the end of the training my model scored 0.0739 as Training loss while in validation set it had 0.0760 as loss




At testing set my model scored 0.045 while the bench mark CNN scored 6.3 as test Mean-Squared-Error

So it seems that model generalize very well on unseen data as it's score on test set which is unseen on training

I tried different shuffles for dataset and the model did same performance

The model can be trusted if we did some improvements to it  as supplying larger datasets with different environments at different lighting conditions and testing it also if we interpreted and visualized the output of convolutional layers and see the saliency objects that the model prediction relies on

# Justification

The Pilotnet Architecture beated the benchmark naïve model with much higher performance with error rates 0.045 :6.3

This is due to the nature of convolutional neural networks . as CNNs have revolutionized pattern recognition field as it automatically learns internal representations and might learn some feature that are not in the criteria of the human

We can further decrease the error of the model by  collecting more new data and by trying different techniques and hyperparameters empirically till we reach the highest score we can get

# Reflections

The process done in this project can be summarized into :

1. Initial problem was proposed
2. A dataset was found
3. Data were preprocessed and split
4. A benchmark model created and evaluated
5. A more robust model was trained to solve the problem
6. The robust model was compared to the benchmark model

# Improvement

- We can improve the model's error rate by feeding more new data
- We can use other architectures as LSTMs that can get benefit from the information previous frames instead of depending on one frame as consecutive frames contain information related to each other and the environment in this time

## Reference


https://www.mcca.com/wp-content/uploads/2018/04/Autonomous-Vehicles.pdf

https://en.wikipedia.org/wiki/History_of_self-driving_cars

https://arxiv.org/pdf/1604.07316.pdf

https://medium.com/@erikshestopal/udacity-behavioral-cloning-using-keras-ff55055a64c

https://github.com/SullyChen/Autopilot-TensorFlow