Department of Computer Science and Engineering

Final Year Project - 2020

Analysis Of AI based Self-Driving Car using DenseNet architecture

Presented by,
Rishav Ghosh , Roll -214
Prerana Chakraborty , Roll -223

Under the guidance of
Prof. Subhashree Roy

Acknowledgement

The given project " Analysis of AI based Self Driving Car using DenseNet Architecture" has been been presented according to the given instructions of our mentor Prof. Subhashri Roy. The subject matter has been explained in a lucid language.

Self driven cars are the rising trend in the field of AI and Automobile Technology. Different companies worldwide are trying out possibilities to enhance the design of these types of cars. Our project deals with the Neural Network that forms the core behind such cars. Different architectures and designs have been proposed. Our project uses the novelty of the recent DenseNet networks. In particular we define a design utilizing the standard concepts of DenseNets and use it as our model. We compare our results with that of the architecture proposed by Nvidia Team in their self driving cars. Experimental results show that our proposed model works better in practice as compared to the one proposed by Nvidia. We also elaborate our process of data generation and the entire training process.

We are indebted to our mentor from whom we have collected the ideas to represent the topic in a different way. Any suggestions would be gratefully acknowledged.

-Rishav Ghosh
University Roll : 10400216082
Regn : 161040110290 of 2016-17

- Prerana Chakraborty
University Roll : 10401616086
Regn : 161040110660 of 2016-17

Table Of Contents

INTRODUCTION 5

LITERATURE STUDY 6

OLD vs NEW 6

METHODOLOGY 9

Planning 10

Simulator Selection 10

Data Collection 12

Setting up Google Colab 13

What is Google Colaboratory? 13

Implementation 13

Implementation of the models 13

Linking the Google Colab file with Google drive 15

Data Preprocessing 15

Reading the training data from Google Drive 15

Data Description 15

Storing the processed data in a separate file 15

Exploratory Data Analysis on the processed data 16

What is Violin plot ? 16

Anatomy of the violin plot 16

Base RMSE of the Steering_Angle, What is RMSE ? 17

Standardization of steering angle values 17

Converting the image paths to numpy array 18

Image Preprocessing 18

Resizing of image 18

Grayscale Conversion 19

Canny Transformation 19

Need for Canny Transformation 20

Noise Reduction 20

Median based Thresholding 20

Using a different choice of Convolution Network 22

Architecture of our model 24

Proposed Algorithm in a Nutshell 26

Testing the models (Experimentation) 28

Results of Implementation 31

Future Works 31

Bibliography 33

1. INTRODUCTION

Self driving cars and their implementations in real life are becoming quite popular with the advent of technology and Deep Learning. A **self-driving car**, also known as an **autonomous vehicle** (**AV**), **connected and autonomous vehicle** (**CAV**), **driverless car**, **robo-car**, or **robotic car** is a vehicle that is capable of sensing its environment and moving safely with little or no human effort.Self-driving cars combine a variety of sensors to perceive their surroundings, such a radar, lidar, sonar, etc.

Experiments have been conducted on automated driving systems (ADS) since at least the 1920s; trials began in the 1950s. The first semi-automated car was developed in 1977, by Japan's Tsukuba Mechanical Engineering Laboratory, which required specially marked streets that were interpreted by two cameras on the vehicle and an analog computer. The vehicle reached speeds up to 30 kilometres per hour (19 mph) with the support of an elevated rail.

A landmark autonomous car appeared in the 1980s, with Carnegie Mellon University's Navlab and ALV projects funded by the United States' Defense Advanced Research Projects Agency (DARPA) [4] starting in 1984 and Mercedes-Benz and Bundeswehr University Munich's EUREKA Prometheus Project [6] in 1987. By 1985, the ALV had demonstrated self-driving speeds on two-lane roads of 31 kilometres per hour (19 mph), with obstacle avoidance added in 1986, and off-road driving in day and nighttime conditions by 1987. A major milestone was achieved in 1995, with CMU's NavLab 5 completing the first autonomous coast-to-coast drive of the United States. Of the 2,849 mi (4,585 km) between Pittsburgh,

Pennsylvania and San Diego, California, 2,797 mi (4,501 km) were autonomous (98.2%), completed with an average speed of 63.8 mph (102.7 km/h). From the 1960s through the second DARPA Grand Challenge [5] in 2005, automated vehicle research in the United States was primarily funded by DARPA, the US Army, and the US Navy, yielding incremental advances in speeds, driving competence in more complex conditions, controls, and sensor systems. Companies and research organizations have developed prototypes.

2. LITERATURE STUDY

We have performed background studies on different self driving cars that have already been implemented in real life. Our primary point of concern was the self driving car that had been recently released by NVIDIA [1], and neural architecture.
We have also taken into account other papers that reveal different simulators for an online training of these cars. Finally we have considered the DenseNet [2] model and its original paper from which we have derived our ideas and the design of our own DenseNet model for training Self-Driving Cars.

2.1. OLD VS. NEW

The objective of our project was to build a model better than that proposed in the Nvidia Paper End to End Learning for Self Driving Cars and succeeded to a good extent. In this section we will talk about the differences in the models.

Data Difference
The training data in both the cases had timestamped images from 3 cameras that were mounted behind the windshield of the data acquisition car and the steering angle. However the Nvidia dataset was augmented with single images

and steering labels to prevent the car from drifting away the road even if there was an error. The steering label for the images is adjusted to one that would steer the vehicle back to the desired location and orientation in two seconds.

Model Difference

The NVidia network consists of 9 layers, including a normalization layer, 5 convolutional layers and 3 fully connected layers. The input image is split into YUV planes and passed to the network.The first layer performs image normalization . Strided convolutions are used in the first 3 layers with 2x2 strides and 5x5 kernel size . Non-strided convolutions are used in the last 2 layers with 3x3 kernel size.The convolution layers are followed by 3 fully connected layers.The network has about 27 million connections and 250 thousand parameters.The network model is shown in the following figure.

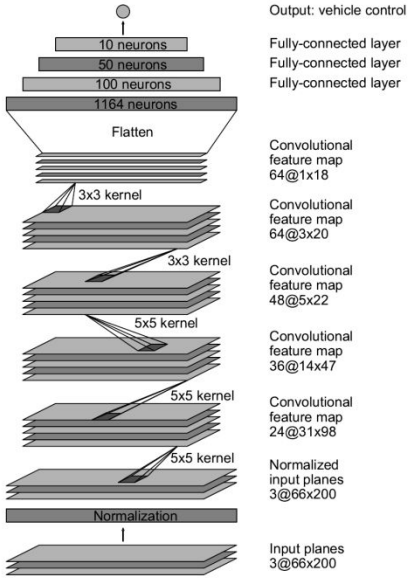


Figure 4: CNN architecture. The network has about 27 million connections and 250 thousand parameters.

In our model we initially use a Transform function . This function is used to preprocess the images before feeding them in the convolution network. The Transform function transforms the RGB images to edge filtered black and white images . This processing could have been done in the CNN model but it would have been time consuming . To make our model faster we preprocessed

them beforehand. After transformation the images are fed into the Densenet model.
We use Densenet architecture to build our model. The model consists of 2 dense blocks separated by a transition layer which is followed by 3 feature reduction layers and 3 fully connected layers.
In each of the dense blocks we have 3 convolution layers with kernel size 5x5 concatenated to each other. The dense blocks are separated by an average pooling layer which is the transition layer .

Our Total params: 347,778
Total Trainable params: 347,372

In particular, Densenet is a widely recognized model however we have modified the original architecture by adding more fully connected layers and countering the increased parameters by using dropout and removing any Global Average Pooling. The architecture works better, consumes less parameters and gives better results which are explained in detail in the next sections.
We implemented both the models in our project and observed the results.

Result Difference

We calculated the RMSE value of the steering angles and accuracy allowing a 5 degree deviation.

	NVIDIA Model	**Densenet Model**
RMSE	0.0307	0.024
Accuracy	89.371%	90.372%

Thus it is observed that our model produced a better RMSE value and higher accuracy than the NVIDIA model.

3. METHODOLOGY

3.1 Introduction

This section will explain in detail the methodology used for completing this project.The methodologies used are referenced from prominent research papers .
The steps are followed to achieve the objective of our project and to accomplish satisfactory results.
The methodology mainly comprises of 3 major steps
1. Planning
2. Implementation
3. Result Analysis

Under planning , there are 3 subsections which include a) Selection of the correct simulator , b) Collecting the data through it and c) Setting up Google Colab for simultaneous implementation.

Under implementation , there are 2 subsections which include a)Implementing the project to achieve the objective and b) Testing our implementation.

Under result analysis , the results of the models are analyzed to conclude which performs better .Below is a visualisation of the methodology used in this project.

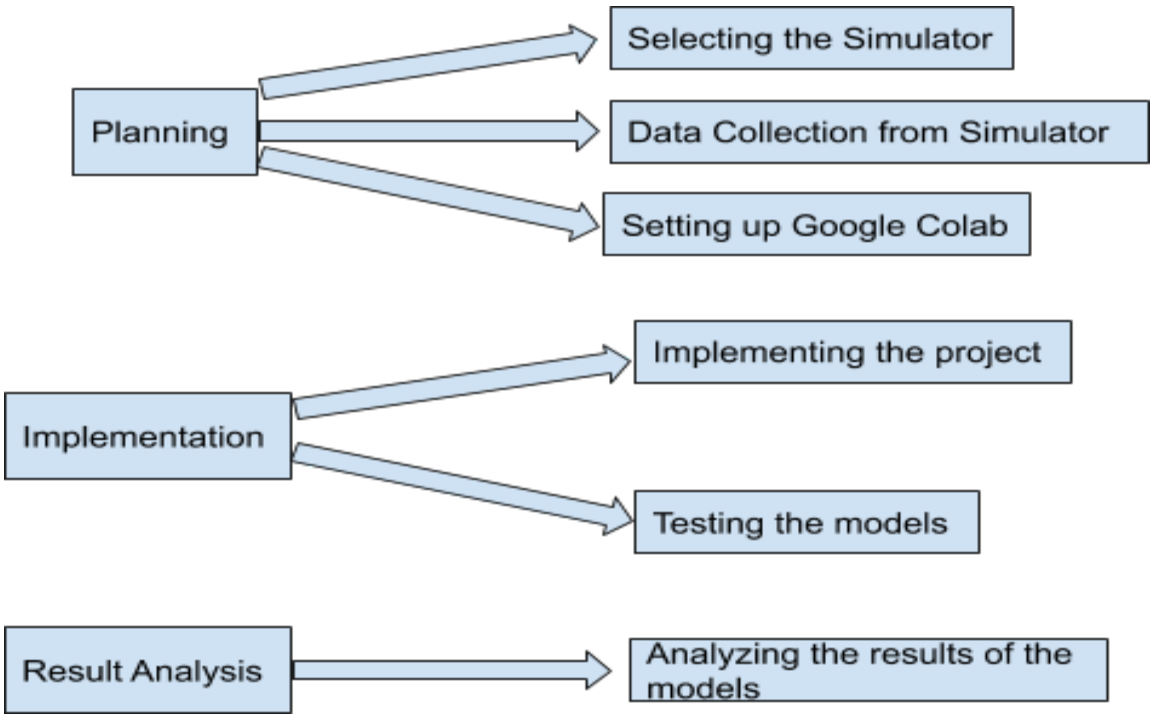


Fig 1 : Approach methodology for our project

Now we will discuss the methodologies in detail.

3.2 Planning

Planning has been done to select the simulator for generating the data . Google Colaboratory has also been set up with a separate Google account for simultaneous access by the project members.

3.2.1. Simulator Selection

Udacity has recently made its Self Driving car simulator Source code available on their Github.
which was originally built to teach their Self-Driving Car Engineer Nanodegree students.
The simulator lets us manually drive the car to generate training data .

Fig 2 : The Udacity Simulator

This simulator is selected because here we can drive the car on a single track , with no opposite cars and without any traffic signal. Also 3 cameras are used for generating the images while driving the car .We planned to implement a simple version of a self driving car and therefore went with this simulator .

3.2.2. Data Collection

The simulator is used to record the training data and images.

Under the **Select Track** option , a track is selected where we want to drive our car.

In **Training Mode ,** we drive the car on the chosen track using the keyboard controls.
To record the data while driving , we press the **'R'** key on the keyboard. To finish recording we again press '**R**'.
While recording , **a dataset** is generated simultaneously in a predefined location alone **center** , **right** and **left** camera images.

C:\Users\ADMIN\Desktop\IMG\IMG\	C:\Users\ADMIN\Desktop\IMG\	C:\Users\ADMIN\Desktop\IM	0	0	0	9.494225
C:\Users\ADMIN\Desktop\IMG\IMG\	C:\Users\ADMIN\Desktop\IMG\	C:\Users\ADMIN\Desktop\IM	0	0	0	9.363687
C:\Users\ADMIN\Desktop\IMG\IMG\	C:\Users\ADMIN\Desktop\IMG\	C:\Users\ADMIN\Desktop\IM	0	0	0	9.253245
C:\Users\ADMIN\Desktop\IMG\IMG\	C:\Users\ADMIN\Desktop\IMG\	C:\Users\ADMIN\Desktop\IM	0	0	0	9.144506
C:\Users\ADMIN\Desktop\IMG\IMG\	C:\Users\ADMIN\Desktop\IMG\	C:\Users\ADMIN\Desktop\IM	0	0	0	9.037059
C:\Users\ADMIN\Desktop\IMG\IMG\	C:\Users\ADMIN\Desktop\IMG\	C:\Users\ADMIN\Desktop\IM	0	0	0	8.930047
C:\Users\ADMIN\Desktop\IMG\IMG\	C:\Users\ADMIN\Desktop\IMG\	C:\Users\ADMIN\Desktop\IM	0	0	0	8.806398
C:\Users\ADMIN\Desktop\IMG\IMG\	C:\Users\ADMIN\Desktop\IMG\	C:\Users\ADMIN\Desktop\IM	0	0	0	8.701837
C:\Users\ADMIN\Desktop\IMG\IMG\	C:\Users\ADMIN\Desktop\IMG\	C:\Users\ADMIN\Desktop\IM	0	0	0	8.598557
C:\Users\ADMIN\Desktop\IMG\IMG\	C:\Users\ADMIN\Desktop\IMG\	C:\Users\ADMIN\Desktop\IM	0	0	0	8.496536
C:\Users\ADMIN\Desktop\IMG\IMG\	C:\Users\ADMIN\Desktop\IMG\	C:\Users\ADMIN\Desktop\IM	0	0	0	8.395335
C:\Users\ADMIN\Desktop\IMG\IMG\	C:\Users\ADMIN\Desktop\IMG\	C:\Users\ADMIN\Desktop\IM	0	0.388051	0	8.908304

Fig :Sample Data

Fig :Center , Left and Right Image at an instant

The training data generated is of size 2.94 MB and the total image folder size is of 652 MB.

3.2.3 Setting up Google Colab

A new Google account is created through which we open a Google Colaboratory File. Also all the necessary files for example the training data and the images were uploaded on the Google Drive of that account.

What is Google Colaboratory?

Colaboratory, or “Colab” for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs. Advantages of using Google Colab:

1. It can be used simultaneously by the project members .
2. It lets us handle huge amounts of data which would have been difficult using our computer resources.
3. It provides us with a single core GPU like Tesla K80 and sufficient amount of RAM.

3.3 Implementation

Our next step involves implementation of the project.It has 2 parts namely implementation of our models and testing the models .

3.3.1 Implementation of the models

This part is further subdivided into the following sections :

- Importing the necessary libraries
- Linking the Google Colab File with the Google drive
- Data Preprocessing
- Image Preprocessing

Each section is described in detail below :

3.3.1.1 Importing the necessary libraries

The most important libraries that have been imported are

- **Numpy** - Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.
- **Matplotlib** - Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.
- **Os**- This module provides a portable way of using operating system dependent functionality. It is mainly used to save necessary intermediate files in the drive .
- **Keras** - Keras is an open-source neural-network library written in Python designed to enable fast experimentation with deep neural networks. It focuses on being user-friendly, modular, and extensible.
- **OpenCV**- OpenCV-Python is a library of Python bindings designed to solve computer vision problems.
- **Pandas**- Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool,built on top of the Python programming language.
- **Seaborn** - Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- **Math** - This module provides access to the mathematical functions defined by the C standard.
- **Sklearn.Preprocessing** - The sklearn.preprocessing package provides several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators.
- **Tensorflow** - TensorFlow is a Python library for fast numerical computing

created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow.

3.3.1.2 Linking the Google Colab file with Google drive

The Google drive is mounted to the Google Colab file so that the training data and images uploaded to Google Drive is available for use in Google Colab.

3.3.1.3 Data Preprocessing

This section is further subdivided into following subsections :

a) Reading the training data from Google Drive

The training data that is uploaded is to the Drive namely "driving_log.csv" is read . The data has the following columns name center, left right , steering_angle , throttle , reverse and speed. It has 14492 rows and 7 columns.

Data Description

Center : contains path for center images
Left : contains path for left images
Right : contains path for right images
steering_angle : Contains steering angle values in radian
throttle : contains boolean value whether throttle was applied or not
reverse : contains boolean value whether reverse gear was applied or not
speed : contains value for speed of the car in km/hr

b) Storing the processed data in a separate file

The original dataset has paths of the images in our computer folder. But now that we are working on Google drive and have the images uploaded in the drive so their location has changed . As a result we need to explicitly change the location in the dataset.

After changing the path of the images in tha dataset, we save it as processed.csv .

c) Exploratory Data Analysis on the processed data

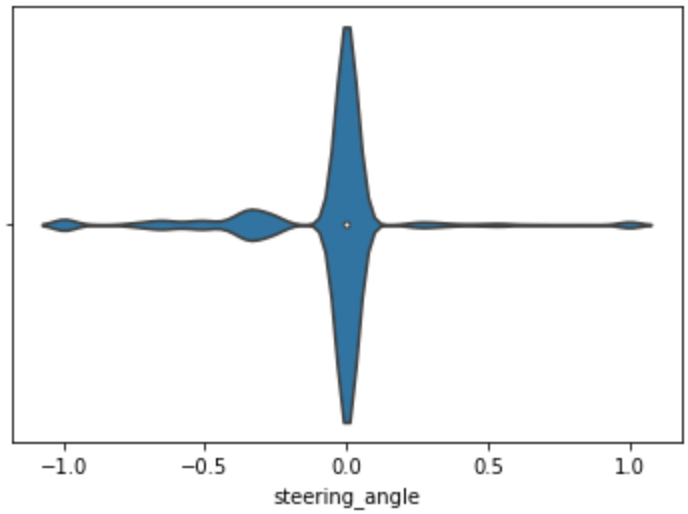
Our goal is to predict the steering angle values from the images , so we visualise the distribution of the steering angles using violin plot.

What is Violin plot ?

A violin plot is a method of plotting numeric data. It is similar to a box plot, with the addition of a rotated kernel density plot on each side.

Violin plots are similar to box plots, except that they also show the probability density of the data at different values

Anatomy of the violin plot



The white dot represents the median.Wider sections of this plot represent a higher probability that members of the population will take on the given value ; the skinnier sections represent a lower probability.This implies that there is a higher probability that the steering angle value is 0. Also that values range from approximately -1 radian to +1 radian .

d) Base RMSE of the

Steering_Angle, What is RMSE ?

Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). The formula is

$$RMSE = \sqrt{(f-o)^2}$$

Where f = expected values

o= observed values

It is mainly used in regression problems.

As we have seen from the distribution of steering angles that most of the values tend to be 0 radian ,we try to have a base RMSE by taking into consideration a possibility that all the predicted values = mean of the steering angle or 0 . This is a worst case scenario . Our objective is to get a RMSE that is significantly less than the base RMSE value.

Base RMSE = $\sqrt{1/n(actual\ steering\ angle - mean(steering\ angle))}$

where n is total number of observation.

e) Standardization of steering angle values

Standardization is a technique of transforming a data to have a mean of 0 and standard deviation of 1 .

$$z = (x - \mu)/n$$

x is data value

μ *is mean of the data*

n is no of observations

Data standardization is about making sure that data is internally consistent; that is, each data type has the same content and format.

We standardize the steering_angle values so that they are normally distributed.After conversion the data is saved as a text file in the drive.

f) Converting the image paths to numpy array

We retrieve each column of the image paths from the dataset and convert them into numpy arrays and save them as .npy files in the Drive.

Once we are done with data preprocessing we move on to image preprocessing.

3.3.1.4 Image Preprocessing

Under this section , images are preprocessed to retrieve the necessary information .This section is further subdivided into subsections . However the subsections are clubbed into a single transform function . The subsections are

a) Resizing of image

Below is a sample image .

Fig 3 : A Sample Image (taken from Center Camera)

In the above image we see that there are lots of unnecessary details which will not contribute towards prediction of steering angle.For example the sky , trees and the bonnet of the car.The most important thing of the image in tha road. So as a result we cropped the image to get only the road .

So we retrieve 60th to 135th row and 0 to 250th column pixels.

b) Grayscale Conversion

We have used the weighted model of grayscale conversion here . Since red color has more wavelength of all the three colors and green is the color that has less wavelength than red color and gives more soothing effect to the eyes.It means that we have to decrease the contribution of red color, and increase the contribution of the green color, and put blue color contribution in between these two.

So the new equation is of the form is:

$$y = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

Using this algorithm we convert our coloured image to grayscale image.Thus now we have only one 1 dimension image instead of 3 dimension image

c) Canny Transformation

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images.

Need for Canny Transformation

Our goal is to predict the steering angles , so for that we need to detect the path or road . As a result , we tried to detect the edges of the road using canny transformation while reducing the noise .

Noise Reduction

First we need to reduce the noise because edge detection techniques are highly sensitive to noise.

One way to get rid of the noise on the image, is by applying Gaussian blur to smooth it. To do so, image convolution technique is applied with a Gaussian Kernel (3x3, 5x5, 7x7 etc…). The kernel size depends on the expected blurring effect. Basically, the smallest the kernel, the less visible is the blur. In our example, we will use a 3x3 Gaussian kernel.

Median based Thresholding

This method simply takes the median of the image and constructs upper and lower thresholds based on the percentage of the median value .

Lower threshold value= max(0, (1-sigma)*median)

Upper threshold value= min(255, (1+sigma)*median)

A lower value of sigma indicates a tighter threshold, whereas a larger value of sigma gives a wider threshold.In our case sigma=0.45 and it gives a considerably good result. Thus after the Canny transformation we get only the edges of the road . Below is an example of the transformation.

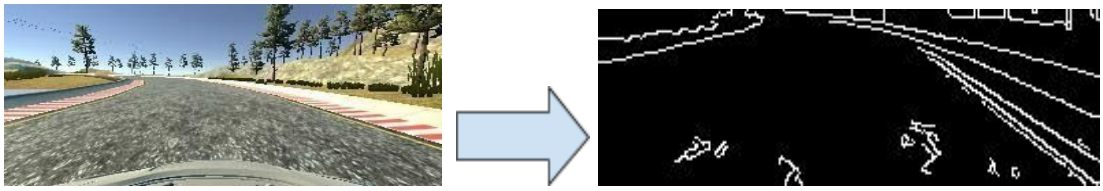


Fig 4: Original Image in RGB (Left) and the corresponding Transformed Image (Right)

Any edges with intensity gradient more than upped threshold value are sure to be edges and those below lower threshold value are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to “sure-edge” pixels, they are considered to be part of edges.Otherwise, they are also discarded.

We have combined the above operations into a single Transform function, and applied to all the image paths of the .npy file to transform the images and save them to separate .npy files in the Drive.

3.3.1.5 Using a different choice of Convolution Network

After the application of the Transform function to all the images of the training dataset, we obtain the edged-filtered 75x250 black and white images which are ready to be fed to our model that we have designed for our objective. The basic architecture is similar to that of a DenseNet Convolutional Network.

DenseNets connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections — one between each layer and its subsequent layer — our network has L(L+1)/ 2 direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters.

Counter-intuitive effect of this dense connectivity pattern is that it requires fewer parameters than traditional convolutional networks, as there is no need to relearn redundant feature maps. Traditional feed-forward architectures can be viewed as algorithms with a state, which is passed on from layer to layer. Each layer reads the state from its preceding layer and writes to the subsequent layer. It changes the state but also passes on information that needs to be preserved.

Definition. **Convolutional -Networks :** The name "convolutional neural network" indicates that the network employs a mathematical operation called *convolution.* Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers. A convolutional neural network is characterized by three primary features – the Kernel(Filter) Matrix (**K**), the spacing or the way of transition of this kernel known as Stride(**S**) and the Padding(**P**) done to keep the shape of the input intact or manipulate as and when necessary.

The formula for calculating the output size for any given convolution layer is

$$O = \frac{(W - K + 2P)}{S} + 1$$

Where, O is the output height/length, W is the input height/length, K is the filter size, P is the padding, and S is the stride.

Definition. **Dense-Blocks:** DenseNets are divided into **Dense-Blocks, where the dimensions of the feature maps remain constant within a block, but the number of filters changes between them.** Each dense block has a number of layers where convolution operation is performed and the output of each layer is concatenated with the output of its preceding layer. A composite transform function is applied to them and the resulting concatenated output is treated as the input of the next layer.

Mathematically,

$$\mathbf{X_l = H_l([X_0, X_1 \ldots . X_{l-1}]),}$$

where, X_l is the final output for layer 'l', '[]' is the concatenation of outputs of previous layers and H_l is a chosen composite function of Batch-Normalization-ReLU and Convolution operation.

The advantage of using this concatenation is that it provides a link from every to another which serves as a path for the gradients update during back-propagation, thereby alleviating the vanishing gradients problem. The size of feature maps(output-filters) remain fixed inside a block and is determined by the parameter **k**, known as the growth rate. It has been seen that a small value of this parameter(say 12 -16) is sufficient for achieving *state-of-the-art* results.

Between a pair of dense blocks is a Transition Layer which performs two sequential operations- first it applies the composite function **H** on the receiving input data volume, in order to reduce the channels to the growth rate **k**. After this, an Average Pooling operation with a kernel size 2x2 and a stride of 2 units is performed and the resulting output is passed to the next dense block. The purpose of the chosen values in the Average Pooling layer is to reduce feature dimensions
to half of its original size. This is a standard value and is chosen in accordance with the standard DenseNet model specified in the original paper.

3.3.1.6 Architecture of our model

The structure of the DenseNet neural network being employed is considerably different from those implemented in the original paper for training on the ImageNet dataset. The model being implemented here is a standard DenseNet, without any bottle-neck or compression attributes. The reason behind such a choice being the nature of the images that are fed into the network (single channel black and white edge-filtered), and hence sufficient accuracy is achieved using a standard architecture ; increasing the number of additional attributes would result in over training and poor performance on the cross-validation set.

Initially, we perform a convolution operation on the input 75x250 image with a kernel size of 5x5, and is padded to keep the feature size same as that of the input. The number of output channels is set to 16. This kernel size and the channels is kept similar to the model used by Nvidia self driving cars. The sole purpose of this Input layer convolution is to increase the number of feature channels (or filters) while keeping the input dimension the same.

This enhanced input is fed to the first dense block. For our objective, we have used two dense blocks, each having three layers within them. In all the three

layers, we have performed the composite operation **H** as mentioned in the previous section, with the same kernel size, then concatenated the output of the current layer with its preceding layers and is sent to the next layer as its input feature. This is done in both the dense blocks. The growth rate (output channels per layer) **k** is kept 12 since a small value of this rate is sufficient for achieving decent data accuracy.

The output of the first dense block is sent to the transition layer. The operations in the transition layer are already explained in the previous section. Since the outputs are concatenated, a 36 channel (12 each for three layers in the dense block) feature is first reduced to 16 channels using the same composite function **H**. This is followed by an Average Pooling operation. The Average Pooling serves the following :

- Reduces the dimension of the feature map to 37x125 which is approximately half of the original size.
- Sets the number of channels to 16 which is the same as set by the input convolution layer before the first dense block.
- Prepares the feature map for the next dense layer in a convenient way.

Similar operations take place in the next dense block. However, the output of this block is treated somewhat differently compared to any original DenseNet model. We have avoided the GlobalAveragePooling operation since it results in loss of sufficient information within the feature maps. Considering the nature and dimension of our original input, adding a GlobalAveragePooling layer in the end results in under-training or a highly biased information from the convolutional network. We define “Feature Reduction” layers, where we first use a series of convolution operations via our composite function **H** to reduce the dimensions and the channels of the obtained feature. We then apply dropout

of 50%, followed by a traditional flattening layer and a series of fully-connected layers. In our context, the combination of flattening and fully connected layers works better since our input itself is not so rich (edged filtered single channel images) hence better predictions are made with such architecture.

The entire **Proposed Algorithm in a Nutshell** can be stated as follows :

Algorithm 1 : Training of AI based Self driven car using modified DenseNet architectures

Step 1: Start

Step 2: Capture the three sets of images of the surroundings

Step 3: Repeat steps 4-6 (Transform function) for all images

Step 4: Crop unnecessary details from the images (any standardized or manual cropping can be applied as long as the turn of the road is captured and the cropping is same for all images)

Step 5: Convert the RGB images into Grayscale images

Step 6: Apply Canny Edge Detection transform to get the black and white edges.

Step 7: Select a random image from the three sets and feed it to the Densenet neural network whose architecture is explained above.

Step 8: Predict the steering angle in radians (allow for some tolerance), use mean squared error as a loss function against the ground truth values and Adam optimizer (with a reduced learning rate of 0.0001, explained in next section)

Step 9: Get the predicted steering angle to drive the car

Step 10: End

The model summary is stated below

Layer Definition	Input Size	Output Size	Operation
Input Layer	75x250x1	75x250x16	Input Convolution
Dense-Block-1	75x250x16	75x250x36	(5x5 CONV + Concat) x3
Transition Layer	75x250x36	37x125x16	(1x1 CONV + 2x2 AvgPool)
Dense-Block-2	37x125x16	37x125x36	(5x5 CONV + Concat) x3
Feature-Reduction-1	37x125x36	19x63x18	5x5 CONV , stride = 2
Feature-Reduction-2	19x63x18	10x32x9	5x5 CONV , stride = 2
Feature-Reduction-3	10x32x9	10x32x9	3x3 CONV , stride = 1
Dropout + Flatten	10x32x9	2880	Dropout(50%) + flatten
FC1	2880	100	Fully-connected + ReLU
FC2	100	50	Fully-connected + ReLU
FC3	50	10	Fully-connected + ReLU

Output	10	1	Linear

In the above table, CONV represents the composite operation **H** of batch-normalization, ReLU activation and the corresponding convolution. The last dimension indicates the number of channels of feature maps. The activation in the final layer is linear since our objective is a regression problem and we consider all real values as an output. A total of 347,372 trainable parameters comprise our proposed model.

4. Testing the models (Experimentation)

We took our dataset and divided the training and test set in 80%:20% ratio respectively. Thereafter we obtained our training set of 11,593 samples and a test set of 2898 samples. Each of these samples is a randomly chosen image from the image generator, 75x250 edged filtered images. For all the experiments, the loss function we had chosen was the Mean Squared Error Loss between the predicted values and the ground truth values since it is a regression oriented problem, and the Adam Optimizer [3] with a reduced learning rate of 0.0001.

4.1 Using our proposed DenseNet architecture based model

We trained our model for 10 epochs keeping the batch-size of 64 samples. Initially we start with a considerably large training loss but with the increase of epochs both the training and validation loss decreases substantially.

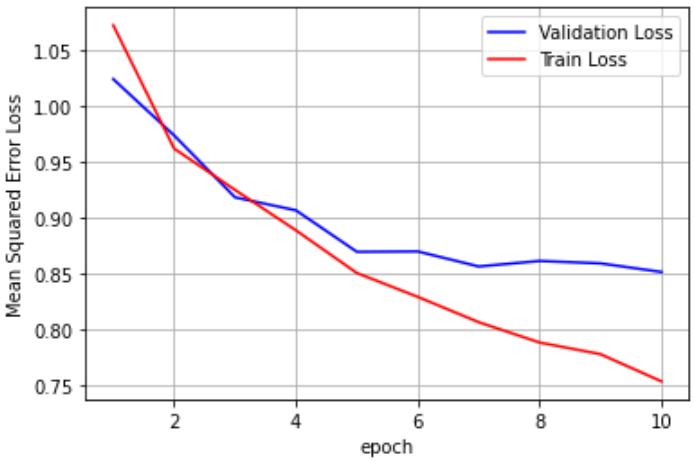


Fig 5: Training and Validation Loss with epochs of our proposed model

We can also visualize the distribution of original and predicted steering angle values as follows

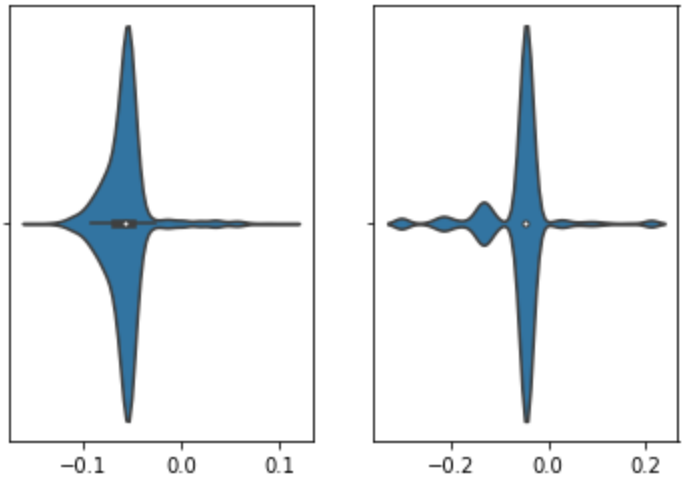


Fig 6 : The Violin Plots for our model

The plot on the left and the right hand side represents the distribution of the predicted and the validation set steering angles respectively.

For calculating the accuracy, we have kept a tolerance of 0.1 radians between the

predicted angles and the steering angles of our test set. We have obtained an accuracy of 90.37 % using the proposed model.

4.2 Using the model proposed by NVIDIA in their self driving car

For the sake of comparison and to verify the performance of our proposed model, we have also trained with the model and architecture as stated by NVIDIA in their paper of self driving car. The primary disadvantage of their convolutional network model is that it comprises more than 4 lakh parameters which is much larger in magnitude compared to our model. All other conditions regarding the loss function, optimizer, epochs and the batch size remains the same.

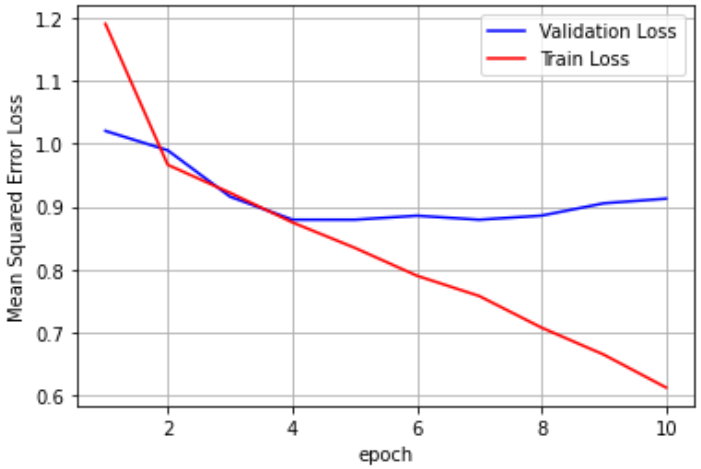


Fig 7: Training and Validation Loss of the model proposed by Nvidia

From the above graph we can see that the training loss is initially higher compared to our DenseNet model at the early stages of optimization. As the optimization progresses the training loss decreases however the validation loss gets stuck to 0.9 and increases henceforth resulting in comparatively poor performance in the validation set.

We kept the same tolerance of 0.1 radians and the same test set for calculating the accuracy of this model. This model provides us with an accuracy of 89.3% .

5. Results of implementation

From the above, we can see that our proposed model to train self-driving cars works well in practice and outperforms famous architectures like the ones proposed by NVIDIA, with fewer parameters. This is more so because of the nature of the dense blocks and the feature maps produced inside them. As stated in the original paper,

- Concatenation of outputs increases the compact nature of the model inside the dense blocks.
- With the increase in compactness, the feature maps become richer and hence

  transfer more information to the next layers.
- It has been seen that with increase in compactness near the input layers and near the output layers, the overall performance of any cnn is enhanced, like in DenseNet that too with comparatively lesser parameters.

6. Future Works

In future the above model can be improved by increasing the accuracy and adding new features like automated speed computation, automatic gear changer , automated brakes . Here we have trained our car on a smooth road without any traffic or any other interruptions. But in the real world that is not the case , so self driving cars should be trained on all types of roads and with traffic and people.

The table below outlines the basic characteristics of each of the five levels of autonomous cars as outlined by the Society of Automotive Engineers (SAE):

Level	Defining Characteristics

Level 0 -- No automation	The driver is responsible for all core driving tasks. However, Level 0 vehicles may still include features like automatic emergency breaking, blind-spot warnings, and lane-departure warnings.
Level 1 -- Driver assistance	Vehicle navigation is controlled by the driver, but driving-assist features like lane centering or adaptive cruise control are included.
Level 2 -- Partial automation	Core vehicle is still controlled by the driver, but the vehicle is capable of using assisted-driving features like lane centering and adaptive cruise control simultaneously.
Level 3 -- Conditional automation	Driver is still required but is not needed to navigate or monitor the environment if certain criteria are met. However, the driver must remain ready to resume control of the vehicle once the conditions permitting ADS are no longer met.
Level 4 -- High automation	The vehicle can carry out all driving functions and does not require that the driver remain ready to take control of navigation. However, the quality of the ADS navigation may decline under certain conditions such as off-road driving or other types of abnormal or hazardous situations. The driver may have the option to control the vehicle.
Level 5 -- Full Automation	The ADS system is advanced enough that the vehicle can carry out all driving functions no matter the conditions. The driver may have the option to control the vehicle.

INFORMATION SOURCE: SOCIETY OF AUTOMOTIVE ENGINEERS.

Different companies worldwide are trying to achieve all levels of automation to achieve a complete automated car .

7. Bibliography

[1] Bojarski et. al., " End to End Learning for Self-Driving Cars", *NVIDIA,* arXiv:1604.07316v1, 2016

[2] Huang, Liu, Maaten, Weinberger, "Densely Connected Convolutional Networks," *CVPR, IEEE Xplore*, pp. 4700–4708, 2016.

[3] Kingma, Ba, " Adam: A Method for Stochastic Optimization ", *3rd International Conference for Learning Representations*, 2015.

[4] Norton, Yanco, Ober, Shane, Skinner, Vice, "Analysis Of Human-Robot Interaction At The DARPA Robotics Challenge Trials," *Journal of Field Robotics 32 (3)*, pp. 420–444, 2015.

[5] Singh, Buehler, Lagnemma, "The 2005 DARPA Grand Race Challenge : The Great Robot Race," *books.google.com*, Springer, 2007.

[6] Xie, Trassoudaine, Alizon, Thonat, Gallice, "Active and Intelligent Sensing of Road Obstacles : Application to the European Eureka- Prometheus Project ," *4th International Conference on Computer Vision* , pp. 616–623, 1993.