

A
PROJECT REPORT
ON
**“STEERING ANGLE PREDICTION FOR SELF-DRIVING
CAR USING CONVOLUTIONAL NEURAL NETWORK”**

SUBMITTED TO
SHIVAJI UNIVERSITY, KOLHAPUR
IN THE PARTIAL FULFILLMENT OF REQUIREMENT
FOR THE AWARD OF DEGREE BACHELOR OF ENGINEERING IN
COMPUTER SCIENCE AND ENGINEERING

SUBMITTED BY

MR. PATEL SHYAM PRAVEENBHAI
MR. PATNI SHUBHAM SUBODH
MR. BHOSALE GOURAV SHAILENDRA
MR. MAKANDAR SAHIL SALIM
MR. SHINDE HRISHIKESH SUNIL
MR. MOMIN FAISAL KHALIL

16ITTRCMPN65
16ITTRCMPN66
16EXTRCMPN68
16EXTRCMPN69
16ITTRCMPN70
14CMPN36

UNDER THE GUIDANCE OF
PROF. U. A. NULI



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
D.K.T.E. SOCIETY'S TEXTILE AND ENGINEERING INSTITUTE,
ICHALKARANJI
(AN AUTONOMOUS INSTITUTE)
ACCREDITED WITH 'A+' GRADE BY NAAC
An ISO 9001:2015 Certified
2018-2019

D.K.T.E.SOCIETY'S TEXTILE AND ENGINEERING INSTITUTE,
ICHALKARANJI
(AN AUTONOMOUS INSTITUTE)
(A+ Grade Accreditation by NAAC)
(ISO 9001:2015 CERTIFIED)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



CERTIFICATE

THIS IS TO CERTIFY THAT, PROJECT WORK ENTITLED

**“Steering Angle Prediction for Self-Driving Car Using
Convolutional Neural Network”**

**IS A BONAFIDE RECORD OF PROJECT WORK CARRIED OUT IN THIS
COLLEGE BY**

**MR. PATEL SHYAM PRAVEENBHAI
MR. PATNI SHUBHAM SUBODH
MR. BHOSALE GOURAV SHAILENDRA
MR. MAKANDAR SAHIL SALIM
MR. SHINDE HRISHIKESH SUNIL
MR. MOMIN FAISAL KHALIL**

**16ITTRCMPN65
16ITTRCMPN66
16EXTRCMPN68
16EXTRCMPN69
16ITTRCMPN70
14CMPN36**

**IS IN THE PARTIAL FULFILLMENT OF AWARD OF DEGREE BACHELOR IN
ENGINEERING IN COMPUTER SCIENCE & ENGINEERING PRESCRIBED BY SHIVAJI
UNIVERSITY, KOLHAPUR FOR THE ACADEMIC YEAR 2018-2019.**

**PROF. U. A. NULI
[PROJECT GUIDE]**

**PROF. (DR.) D. V. KODAVADE
[HOD CSE DEPT.]**

**EXAMINER
[EXTERNAL]**

**PROF. (DR.) P. V. KADOLE
[DIRECTOR]**

DECLARATION

We hereby declare that, the project work report entitled “Steering Angle Prediction for Self-Driving Car Using Convolutional Neural Network” which is being submitted to D.K.T.E. Society’s Textile and Engineering Institute Ichalkaranji, affiliated to Shivaji University, Kolhapur is in partial fulfillment of degree B.E. (CSE). It is a bonafide report of the work carried out by us. The material contained in this report has not been submitted to any university or institution for the award of any degree. Further, we declare that we have not violated any of the provisions under Copyright and Piracy / Cyber / IPR Act amended from time to time.

Name	Roll No	Signature
MR. PATEL SHYAM PRAVEENBHAI	16ITTRCMPN65	
MR. PATNI SHUBHAM SUBODH	16ITTRCMPN66	
MR. BHOSALE GOURAV SHAILENDRA	16EXTRCMPN68	
MR. MAKANDAR SAHIL SALIM	16EXTRCMPN69	
MR. SHINDE HRISHIKESH SUNIL	16ITTRCMPN70	
MR. MOMIN FAISAL KHALIL	14CMPN36	

ACKNOWLEDGEMENT

With great pleasure, we wish to express our deep sense of gratitude to Prof. U. A. Nuli for his valuable guidance, support and encouragement in completion of this project report.

In addition, we would like to take opportunity to thank our head of department Dr. D. V. Kodavade for his co-operation in preparing this project report. We feel gratified to record our cordial thanks to other staff members of Computer Science and Engineering Department for their support, help and assistance, which they extended as and when required.

Thank you,

MR. PATEL SHYAM PRAVEENBHAI
MR. PATNI SHUBHAM SUBODH
MR. BHOSALE GOURAV SHAILENDRA
MR. MAKANDAR SAHIL SALIM
MR. SHINDE HRISHIKESH SUNIL
MR. MOMIN FAISAL KHALIL

16ITTRCMPN65
16ITTRCMPN66
16EXTRCMPN68
16EXTRCMPN69
16ITTRCMPN70
14CMPN36

ABSTRACT

Most of the car accidents happen because of human errors while driving. Therefore, we think this is the serious real life problem and this needs to be solved.

In this project, we focused on designing and implementing an efficient model using convolutional neural network for self-driving car module.

Our project entirely focuses on predicting most accurate rotation angle for the cars based on the input image feed. The calculation supports both types of images taken on highway or on local roads. We implement the models for predicting steering angles on the Udacity dataset and visualize the results by using the dataset given by comma ai.

For better performance, our system can run on the GPU using CUDA to process more and more frames to give accurate results quickly. CUDA enables increase in computing performance by harnessing power of GPU.

INDEX

1. Introduction	1
a. Problem definition	2
b. Aim and objective of the project	2
c. Scope and limitation of the project	2
d. Timeline of the project	3
e. Project Cost	4
2. Literature overview	5
a. Literature overview	6
3. Requirement analysis	7
4. System design	10
a. Architecture Design	11
b. Algorithmic description of each modules	13
c. System Modeling	20
1. Block Diagram	20
2. Dataflow Diagram	21
3. Activity Diagram	22
5. Implementation	24
6. Integration and Testing	30
7. Performance Analysis	32
8. Applications	34
9. Installation Guide and User Manual	36
10. Ethics	49
11. References	51

INTRODUCTION

a. Problem Definition

To predict accurate steering angle for advance driver assistance system.

b. Aim and Objective of the Project

The overall aim of the project is to design a system to predict accurate steering angle irrespective of road and weather condition and assist the driver.

The system is proposed to have the following objectives.

1. System should predict the steering angle in real time.
2. System should predict the steering angle irrespective of road condition.
3. System should predict the steering angle irrespective of weather condition.

c. Scope and limitation of the project

Scope:

1. The proposed system will run on GPU.
2. The system can be a part of advanced driver assistance system.
3. In future, system will steer the vehicle automatically.

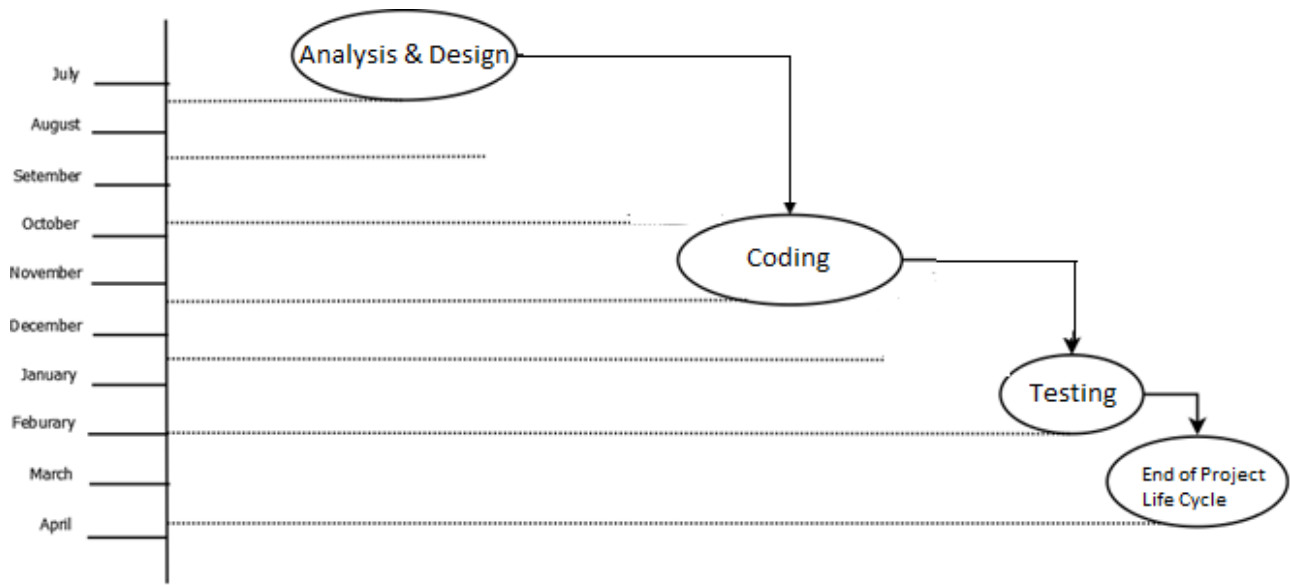
Limitation:

1. The system is designed specifically for constant speed.
2. The system does not work when the vehicle is overtaking.

d. Project Timeline

We have used classic life cycle paradigm also called “Water-Fall Model”. For software engineering which is sequential approach for software development that begins at the system level and progress through analysis, design, coding, testing and maintenance.

We completed the analysis and design phase in November 2018. The coding part was completed upto February 2019 and the testing part was completed in March 2019.



e. Project Cost**Hardware Cost:**

Sr. No	Component	Quantity*price	Price
1	Computer System	1*35000	35000
Total			35000

Cost estimation based on COCOMO model

In this project, the Cost Estimation based on COCOMO (Constructive Cost Model) the formula for this Model is follows:

$$\text{Effort} = \text{Constant} * (\text{Size})^{\text{scale factor}} * \text{Effort Multiplier}$$

- Effort in terms of person-months
- Constant : 2.45 in 1998 based on Organic Mode
- Size : Estimated size in KLOC
- Scale Factor : Combined process factors
- Effort Multiplier (EM) : Combined effort factors

The basic COCOMO equations takes the form:

- Effort Applied (E) = $a_b (\text{KLOC})^{b_b}$ [man-months]
- Development Time (D) = $c_b (\text{Effort Applied})^{d_b}$ [months]
- People required (P) = Effort Applied / Development Time [count]

Where, KLOC is the estimated number of delivered lines (expressed in thousands) of code for project,

The coefficients a_b , b_b , c_b and d_b are given in the following table:

Software Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38

Organic Mode

$$\text{Effort Applied (E)} = 3.0 * (0.264)^{1.12} = 0.68 \text{ Man-months}$$

$$\text{Development Time (D)} = 2.5 * (0.68)^{0.35} = 2.2 \text{ months}$$

$$\text{People required (P)} = 0.68 / 2.2 = 0.3 \sim 1 \text{ People}$$

$$\text{Productivity: } \frac{LOC}{E} = 338 = \frac{LOC}{MM}$$

Literature Overview

Literature Overview

Investigation of current Project and Related Work:

1. NVidia: End to End Learning for Self-Driving Cars:

The organization trained a convolutional neural network (CNN) to map raw pixels from a single front-facing camera directly to steering commands. This end-to-end approach proved surprisingly powerful. With minimum training data from humans, 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 such as in parking lots and on unpaved roads.

2. Deep Learning for Video Classification and Captioning:

The latest developments discussed above have demonstrated the effectiveness of deep learning for video classification. However, current deep learning approaches for video classification usually resort to popular deep models in image and speech domain. The complicated nature of video data, containing abundant spatial, temporal, and acoustic clues, makes off-the-shelf deep models insufficient for video related tasks. This highlights the need for a tailored network to effectively capture spatial and acoustic information, and most importantly to model temporal dynamics. In addition, training CNN/LSTM models requires manual labels that are usually expensive and time-consuming to acquire, and hence one promising direction is to make full utilization of the substantial amounts of unlabeled video data and rich contextual clues to derive better video representations.

3. Visualizing and Understanding Convolutional Networks:

Large Convolutional Network models have recently demonstrated impressive classification performance on the ImageNet benchmark. However, there is no clear understanding of why they perform so well, or how they might be improved. In this paper, we explore both issues. We introduce a novel visualization technique that gives insight into the function of intermediate feature layers and the operation of the classifier. Used in a diagnostic role, these visualizations allow us to find model architectures that outperform Krizhevsky et al. on the ImageNet classification benchmark. We also perform an ablation study to discover the performance contribution from different model layers.

Requirement Analysis

Input: Video frames by camera.

Video captured by front facing camera mounted on dashboard.

Output: Steering Angle

Steering angle predicted using trained CNN model.

Function Requirement:

i. Capturing Videos

Video is nothing but the series of consecutive frames. The camera should capture the video properly.

ii. Convolution

As system gets trained by CNN (Convolutional Neural Network) we need to convolve each filter with the input image (dot product).

iii. Activation function / optimizer

The activation function is actually used to convert each score from the convolutional layer to obtain an optimal solution and hence it also called as optimizer.

Our system uses ReLu – rectifier Linear Unit activation function which is mathematically represented as $\max(0, x)$ where x is the calculated score.

x if $x > 0$

0 if $x \leq 0$

iv. Pooling

The main objective of pooling in our system is to down sample the image. We are using “Max pooling”. It actually selects the maximum score from the each 2×2 block from $n \times n$ tensor with stride of 2.

Max pooling reduces the complexity of the network. We are using max pooling before optimizer because of that optimizer will make the network more optimistic.

Software and Hardware Requirement:

Hardware Requirement:

- RAM – 8 GB
- GPU - NVIDIA GEFORCE GTX 1050
- Operating System - Linux Ubuntu 16.04
- Hard disk size - 1 TB

Software Requirement:

- Python
- Keras
- Tensor Flow on the backend
- anaconda
- cv2
- CUDA

Performance Requirements:

1. The number of terminals to be supported: one terminal i.e. Camera
2. The number of simultaneous users to be supported: single user only driver.
3. Amount and type of information to be handled: we are taking live video streaming of road as an input to our system.

Software System Attributes:

- **Reliability**

Camera will capture video continuously and parallel processing of data will take place with the help of system.

- **Availability**

System will send the input like speed and torque. Based on that, the system will predict the result.

- **Security**

System will be placed in vehicle at proper position and continuous processing will take place without interference of unwanted external environment.

- **Maintainability**

The parallel processing in CUDA platform with the help of python will make the system maintainable.

- **Portability**

NVIDIA GeForce GTX 1050 is capable of supporting multiple cameras through a variety of interface.

System Design

a. Architecture Design:

Architecture Design of Steering Angle Prediction Using CNN:

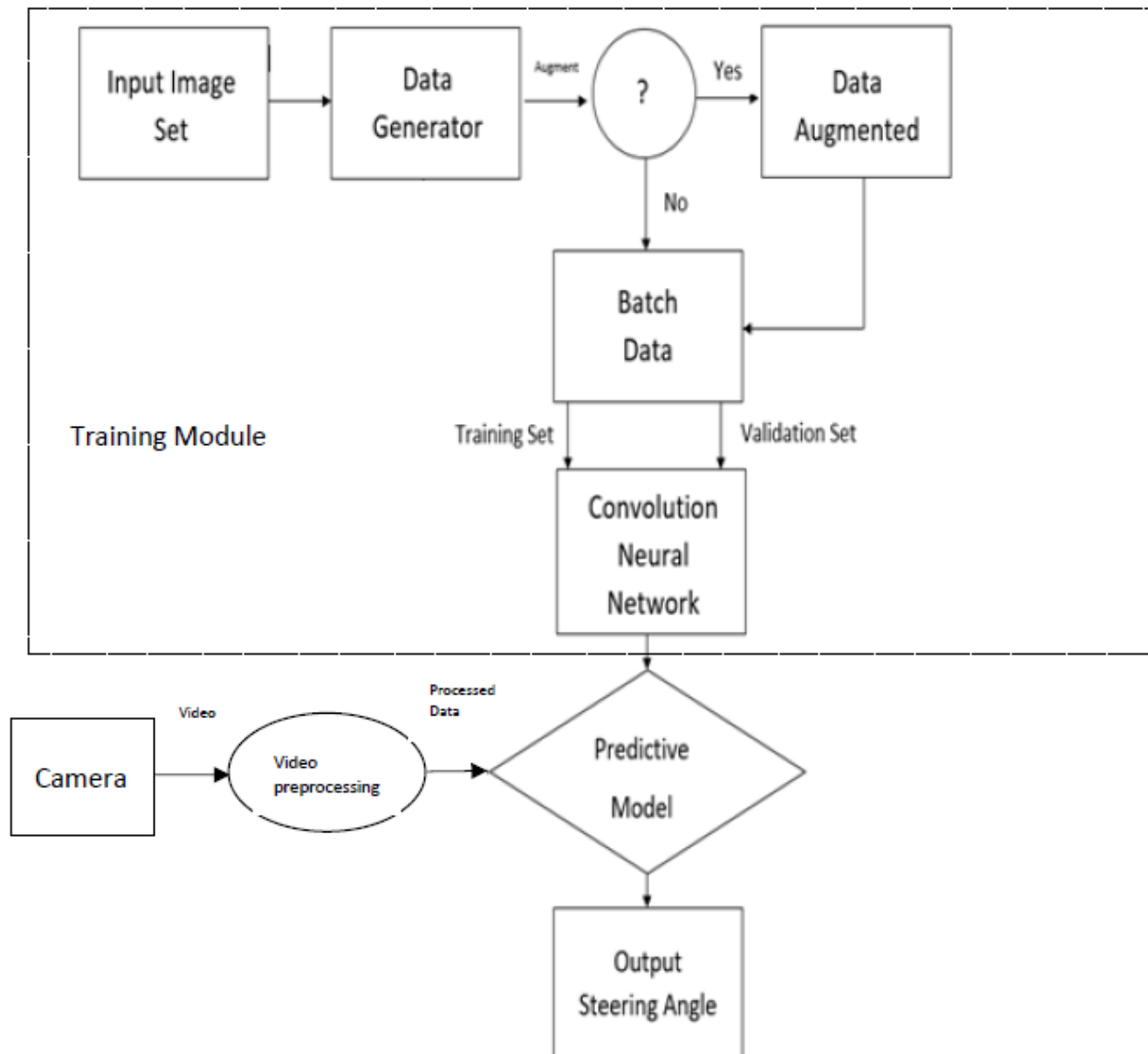


Fig 1. Architectural Design

i. Data Set for training

Input Dataset consist of images taken from left, right and central mounted camera of a car used for driving to capture images. Each image in the set can be identified against unique frame ID. Two CSV files : camera.csv and steering.csv are also provided. Camera.csv contains information about timestamp, width, height, frame_id, filename. Steering.csv contains information about timestamp, angle, torque, speed. Size of all images provided are 640X480. The dataset consists of 4 GB data of images in PNG/ROSBAG format, which is used for training the model and 1.9 GB test data for testing the model.

Dataset contains following types of images:

I.Divided multi-lane highway with heavy traffic.

II. Guardrail and two-lane highway.

III. Divided segment highway images roundtrip

ii. System Architecture

Our system consists of following modules:

Input Module:

Input module reads the images from input dataset directory along with other user specific arguments. Along with reading images, two separate csv file: camera.csv and steering.csv are also read. It also split the input set into two sets : testing set and validation set.

Data Generator:

Data Generator generated various computational specific parameter such as timestamp ranges, shuffle etc. with reference to input set. It calls data augementer with probability of 33% i.e. one of every three input image is passed for data augmentation. At end, batches of image vector formed of original images and augmented images are created.

Data Augmenter:

In every call, data augementer performs different types of augmentation with equal probability i.e. 25 %. Different types of augmentation performed are:

1. Image Flipping
2. Image Rotation
3. Image Blurring
4. Image Sharpening.

Augmented images are added into current image set and their corresponding steering angles are adjusted with respect to augmentation performed.

Convolution Neural Network:

This is core processing module of the system which is trained against the training set received from data generator and weight adjustment are performed with validation set. At last, Predict Model is generated, which will be used to predict the accurate steering angle for each image in testing set.

Output Module:

Output Module simply takes the steering angles generated by predictive model, creates pair of frame_id and steering angle, and writes it into output result files. Accuracy of predicted steering angle is also evaluated using Root Mean Square Deviation.

b. Algorithmic Description:

- **To train the CNN**

1. Read the image from dataset with camera.csv and steering.csv files.
2. Augment the data with 33% probability.
3. Passing the image to CNN model.

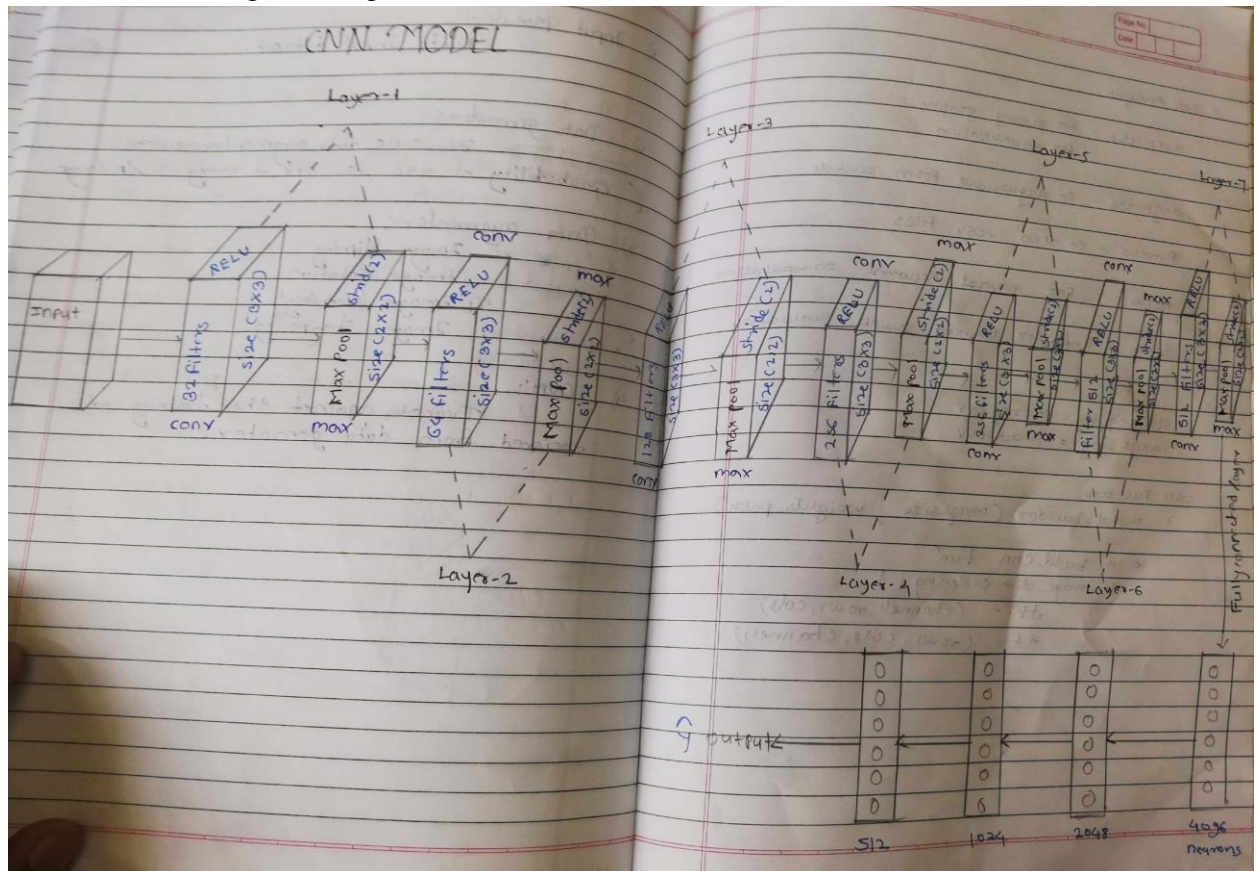


Fig 2. CNN model

4. Weight adjustment using back propagation
5. End of process

- **To test the model**

1. Read the video frames.
2. Give it to pre-trained CNN model.
3. Predict the steering angle for the frame

Architecture of the CNN:

Input:

Image – Road image captured by front facing dash-cam.

Dimensions: 640*480*3

Number of Pixels: 640*480 = 30720

Channels: 3 (RGB)

Number of layers: 11

1. Convolutional Layer 1 + maxpool layer + 0.25 dropout
2. Convolutional Layer 2 + maxpool layer + 0.25 dropout
3. Convolutional Layer 3 + maxpool layer + 0.25 dropout
4. Convolutional Layer 4 + maxpool layer + 0.5 dropout
5. Convolutional Layer 5 + maxpool layer + 0.5 dropout
6. Convolutional Layer 6 + maxpool layer + 0.5 dropout
7. Convolutional Layer 7 + maxpool layer + 0.5 dropout
8. Dense Layer 1 + 0.75 dropout
9. Dense Layer 2 + 0.75 dropout
10. Dense Layer 3 + 0.75 dropout
11. Dense Layer 4 + 0.75 dropout

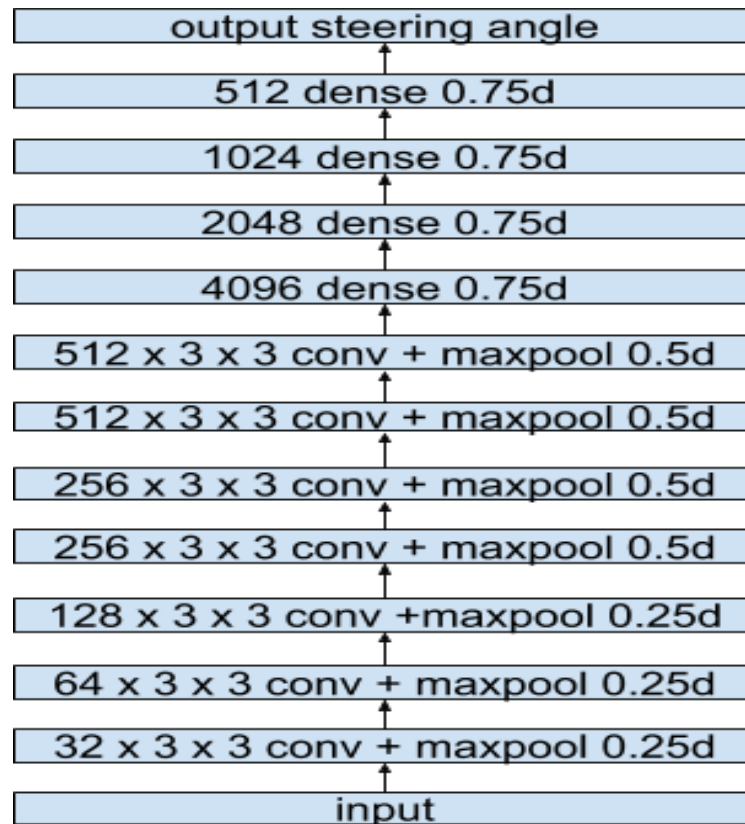


Fig 3. CNN Architecture

Convolutional layer:

Computational Overview

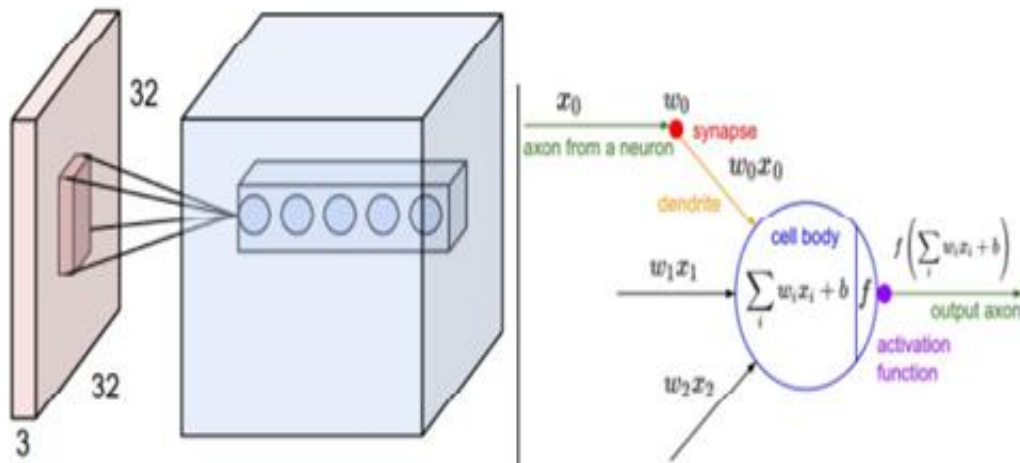


Fig 4. Mathematical Model of Single Perceptron

Convolution:

- The overall concept is to slide the filter over the input image.
- If the input image has three channels, then the filter must have three channels and thus each filter channel has slide over the corresponding channel of input image.
- The filter slides over the image with certain 'Stride'.

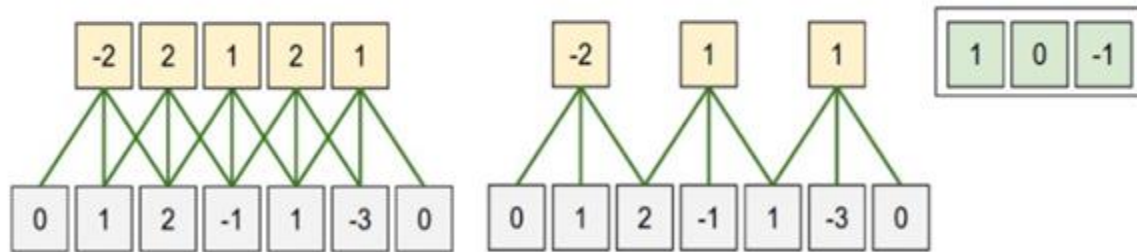


Fig 5. Single Convolution Overview

- Then the corresponding matrix of the image and the filter gets stretched in a single column respectively and it takes dot product.
- Each dot product is resulting into corresponding pixel value of the output image.
- The size of output image is, $O = (I - F + 2P)/S + 1$.

I = input image size, F = Filter Size,

P = Zero padding, S = Stride

e.g. $(33 - 3 + 2(1)) / 2 + 1 = 17$.

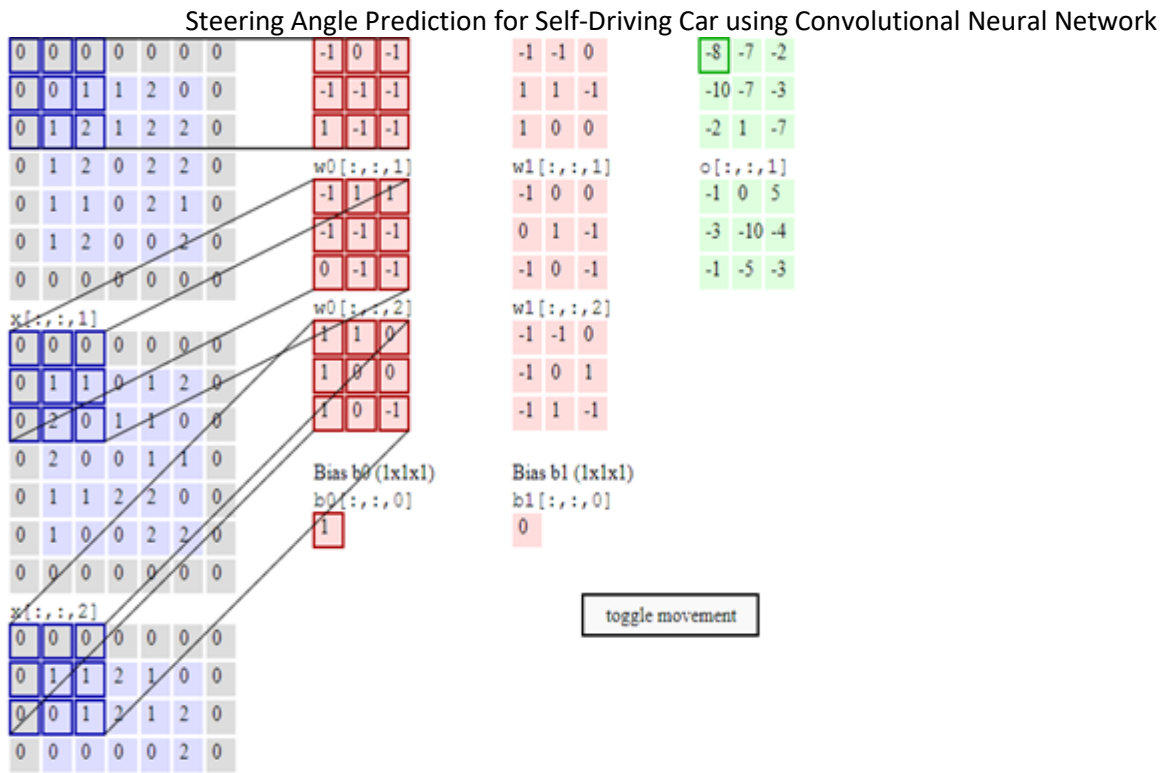


Fig 6. Multiple Convolution Overview

Pooling Computation:

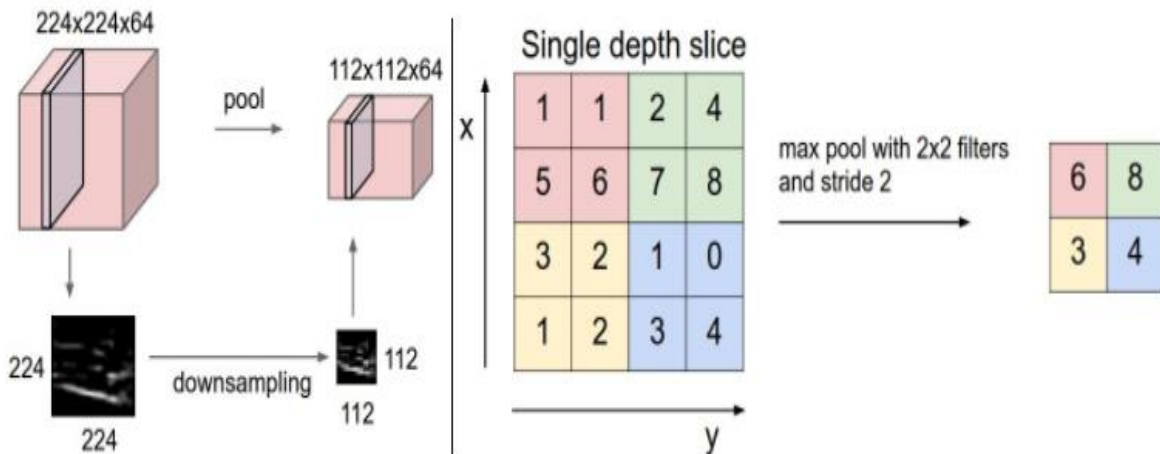


Fig 7. Max-Pooling Overview

Max pooling:

It is the main algorithm, which we are going to use in pooling layer.

Why Max pooling?

It slides an empty filter over the image. E.g., 2*2 dimensional filter with stride of 2 and picks the Maximum value from that matrix. Hence, it reduces the size of input tensor (image) from previous layer and hence it down-samples the input image. Thus, it reduces the computational complexity and reduced tensor in given to the next convolutional layer or FC layer for computation.

Optimization:

ReLU:

The ReLU layer applies an element-wise non-linear activation function to increase nonlinearity in the model.

We know that the value of the neuron in next layer is calculated as follows:

$$\mathbf{X} = \mathbf{x}_1^T \mathbf{w}_1 + \mathbf{x}_2^T \mathbf{w}_2 + \mathbf{x}_3^T \mathbf{w}_3 + \dots + \mathbf{x}_n^T \mathbf{w}_n.$$

Rectifier linear unit is an optimizer which applies a formula **max (0, X)**. That is it removes all negative value and only focus on the value, which will cause neuron fire.

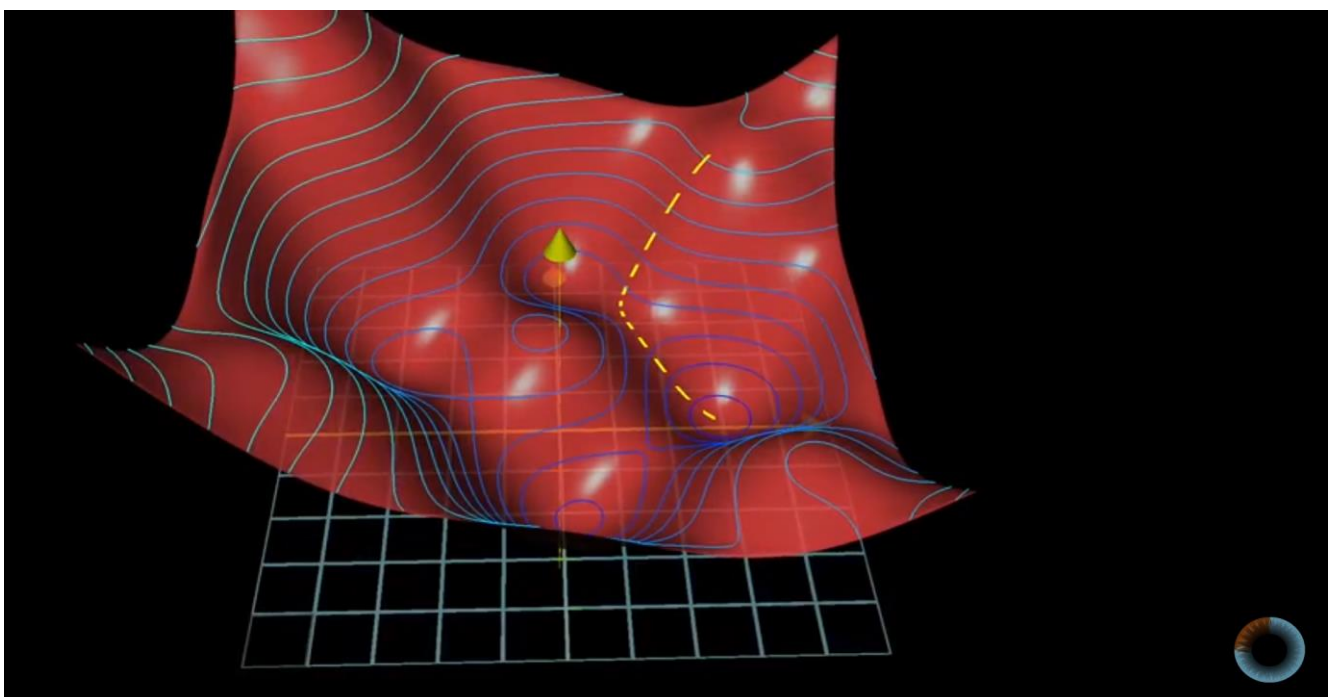
Fully Connected layer:

Fully-connected layers, as the name suggests, is like ordinary NN where each neuron is connected to all The outputs from the previous layer. The last FC layer computes probabilities for each class. For multi-class classification, softmax is a popular choice.

Softmax regression has the following log-likelihood function:

$$l(\theta) = \sum_{i=1}^m \log \prod_{l=1}^k \left(\frac{\exp(\theta_l^T x^{(i)})}{\sum_{j=1}^k \exp(\theta_j^T x^{(i)})} \right)^{1_{\{y^{(i)}=l\}}}$$

Backpropagation:

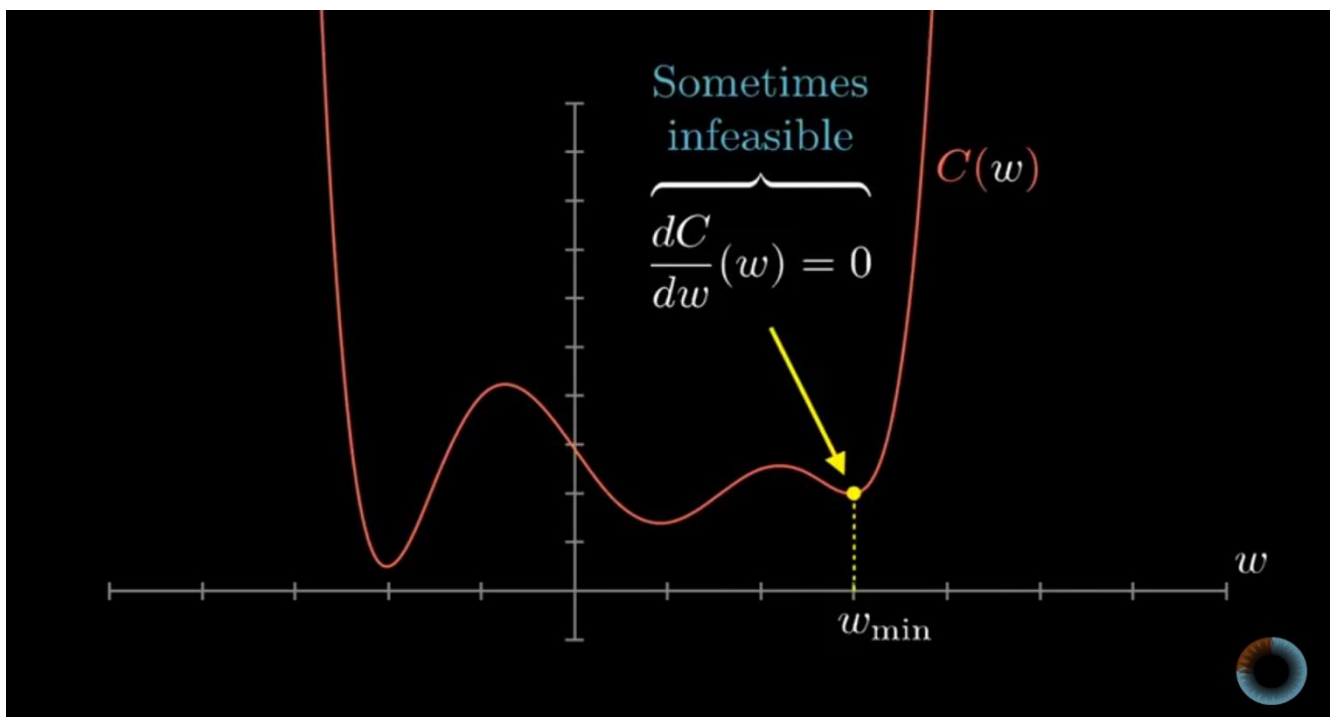
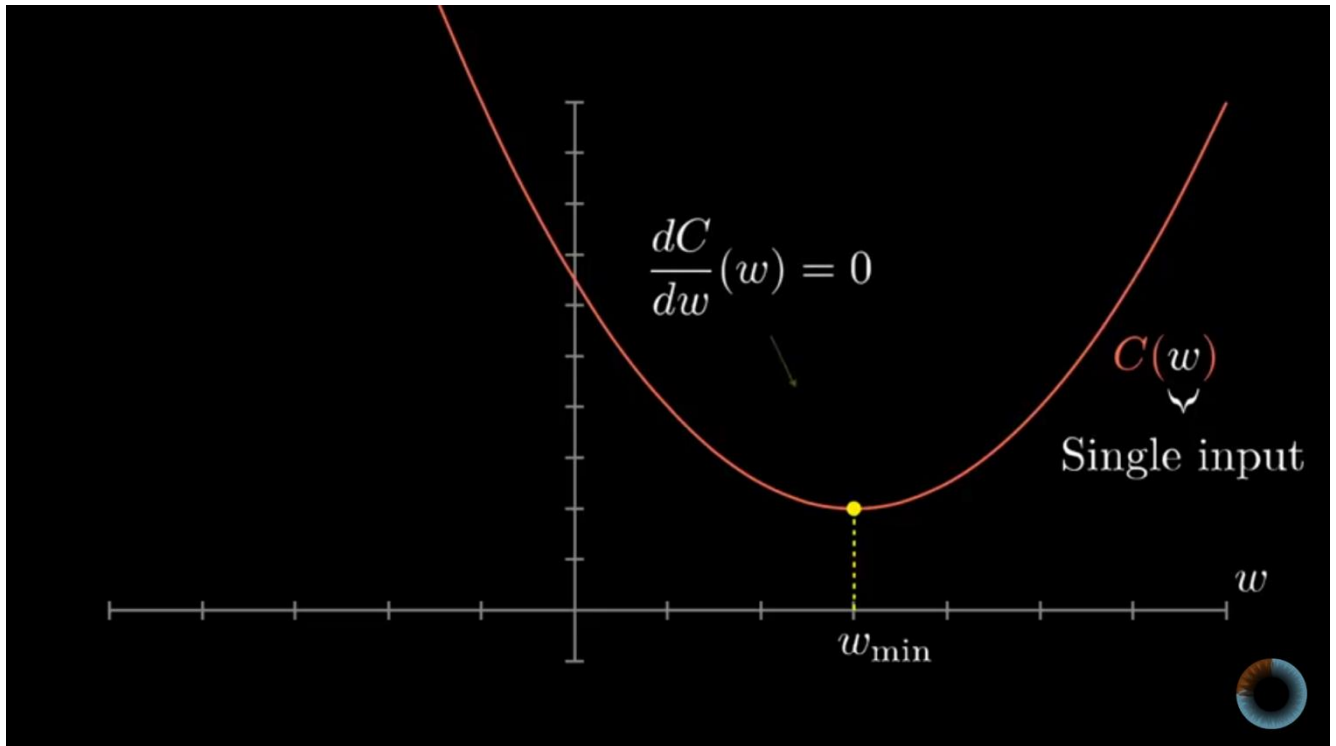


Steering Angle Prediction for Self-Driving Car using Convolutional Neural Network

In fully connected layer by applying softmax function, we predict score (probability) for each class. It is essential that only one neuron out of ten should be fired but at first time, multiple neurons may get fired that is there must be an error in score. So we find the average cost for each class to determine the overall cost of our network.

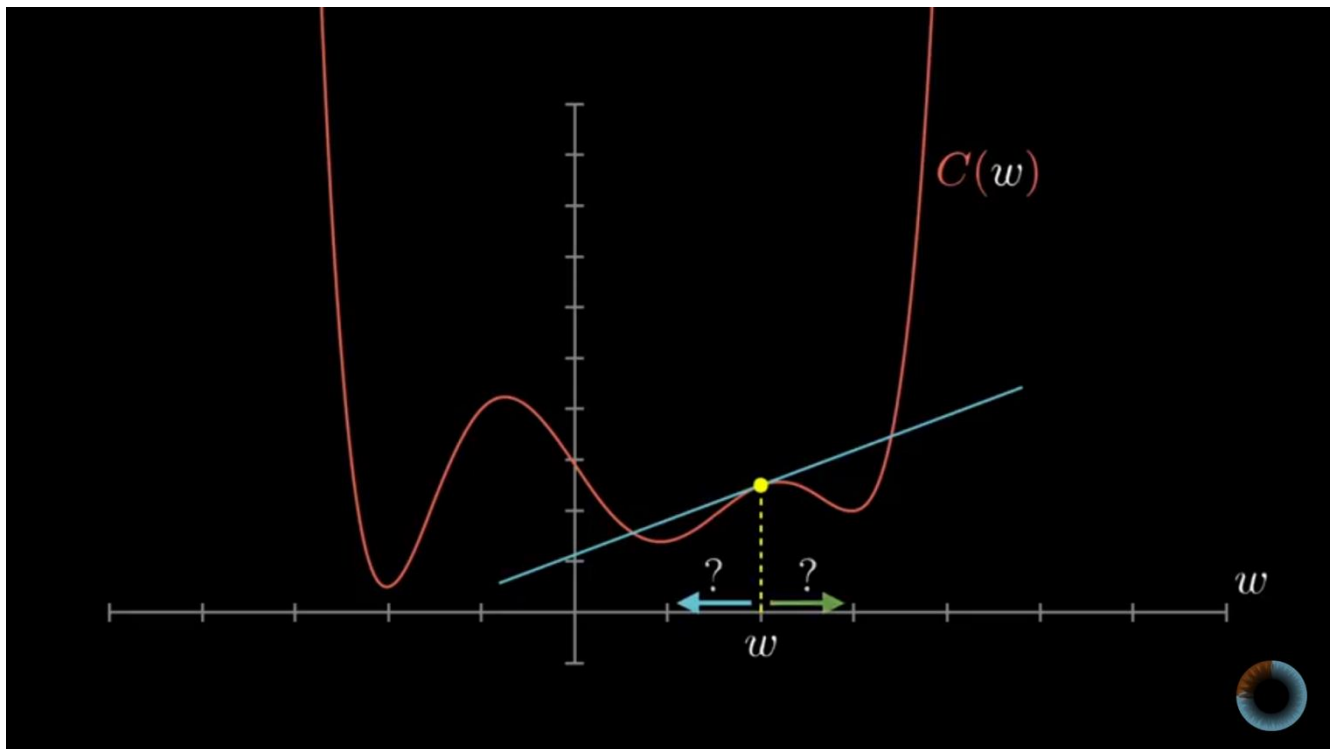
Then we backpropagate to adjust weights and biases. All negative weights, which do not have an impact to fire a respective neuron, will get decreased and all positive weights, which caused the neuron to fire, will get increased.

So, we need to find a minimum function which will reduce the cost i.e. local minima.



Steering Angle Prediction for Self-Driving Car using Convolutional Neural Network

We can start from any input and find the slope of that point. If the slope is positive then we will move to left and to right if slope is negative. We will do this until we find the local minima.



The Backpropagation is the algorithm for determining how a single training example would like to nudge the weights and biases not just in terms of whether they go up or down but in terms of what relative proportions to those changes the most rapid decrease to cost.

c. System Modeling:

1.1 Block Diagram of system

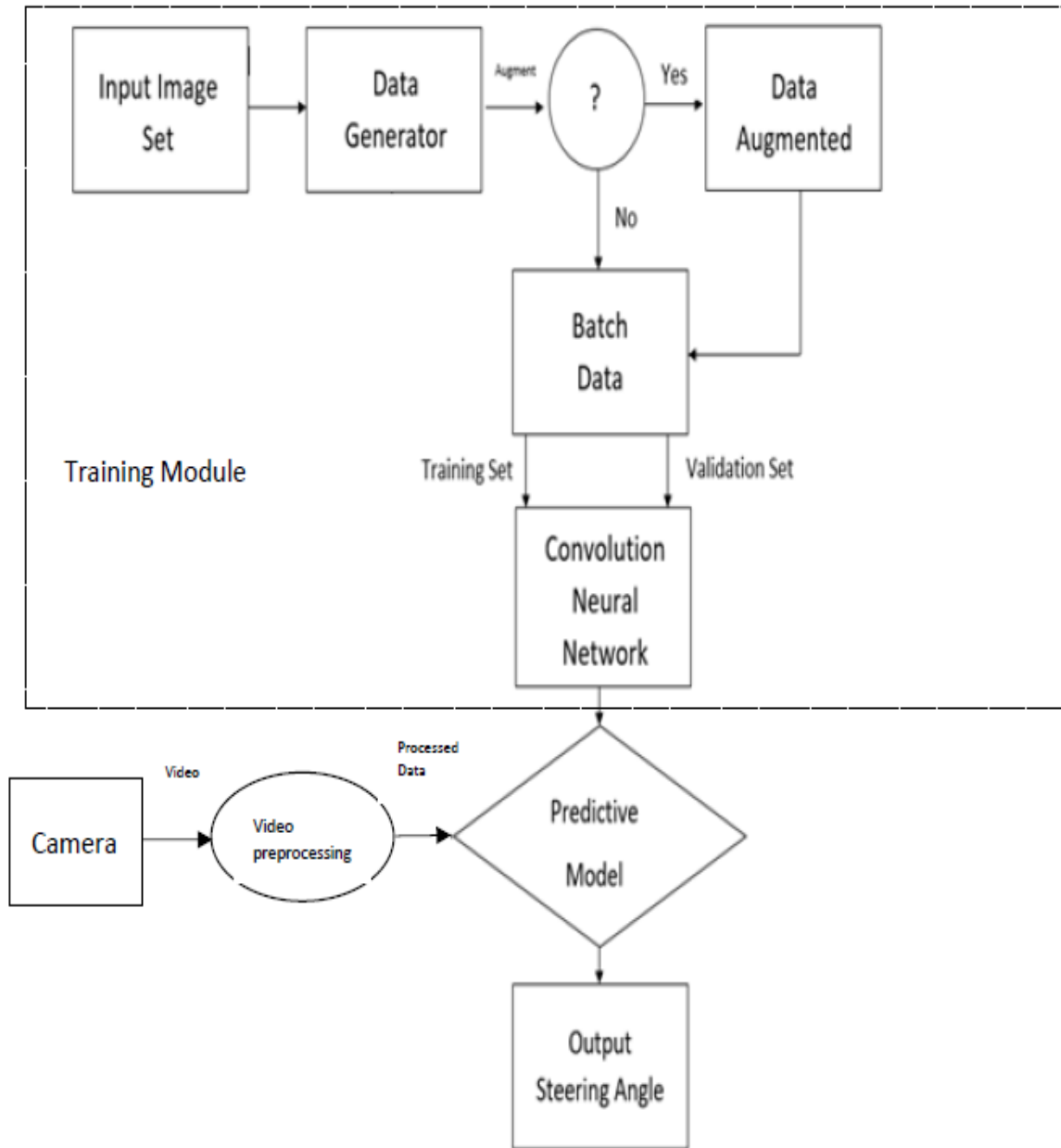
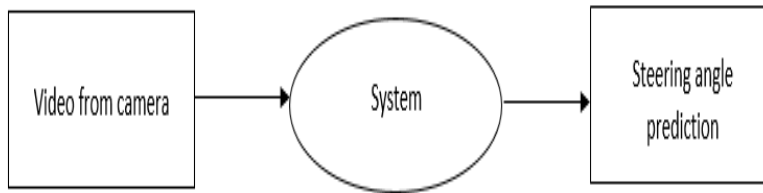


Fig 8. Steering Angle Prediction Block Diagram

2. Data Flow Diagram of System:

DFD LEVEL 0:



DFD LEVEL 1:

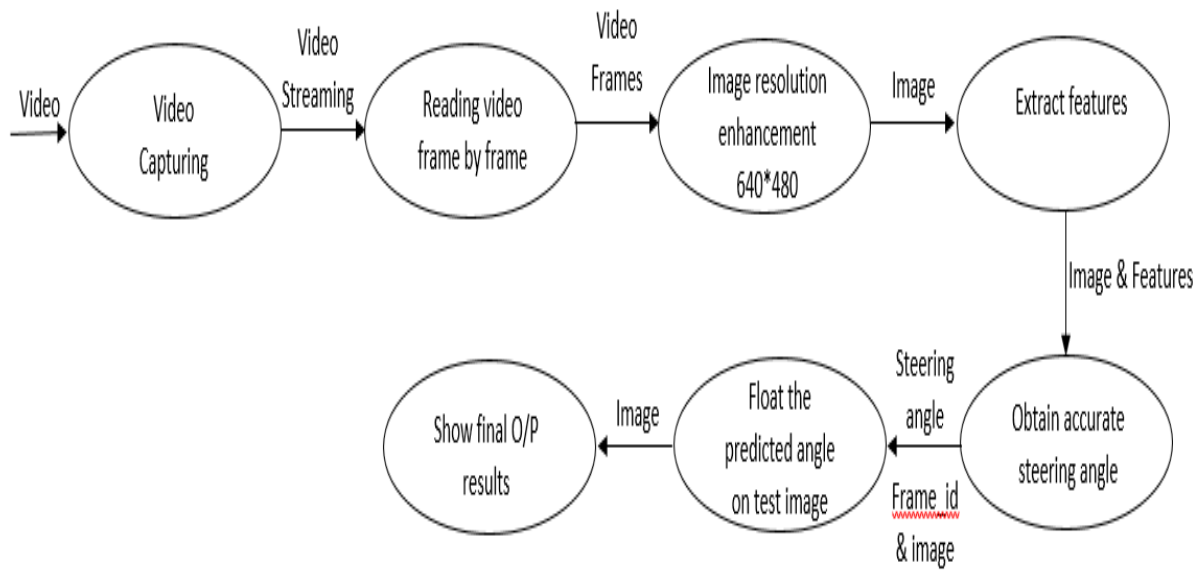


Fig 9. Level 0 & Level 1 Data Flow Diagram of System

3.1 Activity Diagram for Training Model:

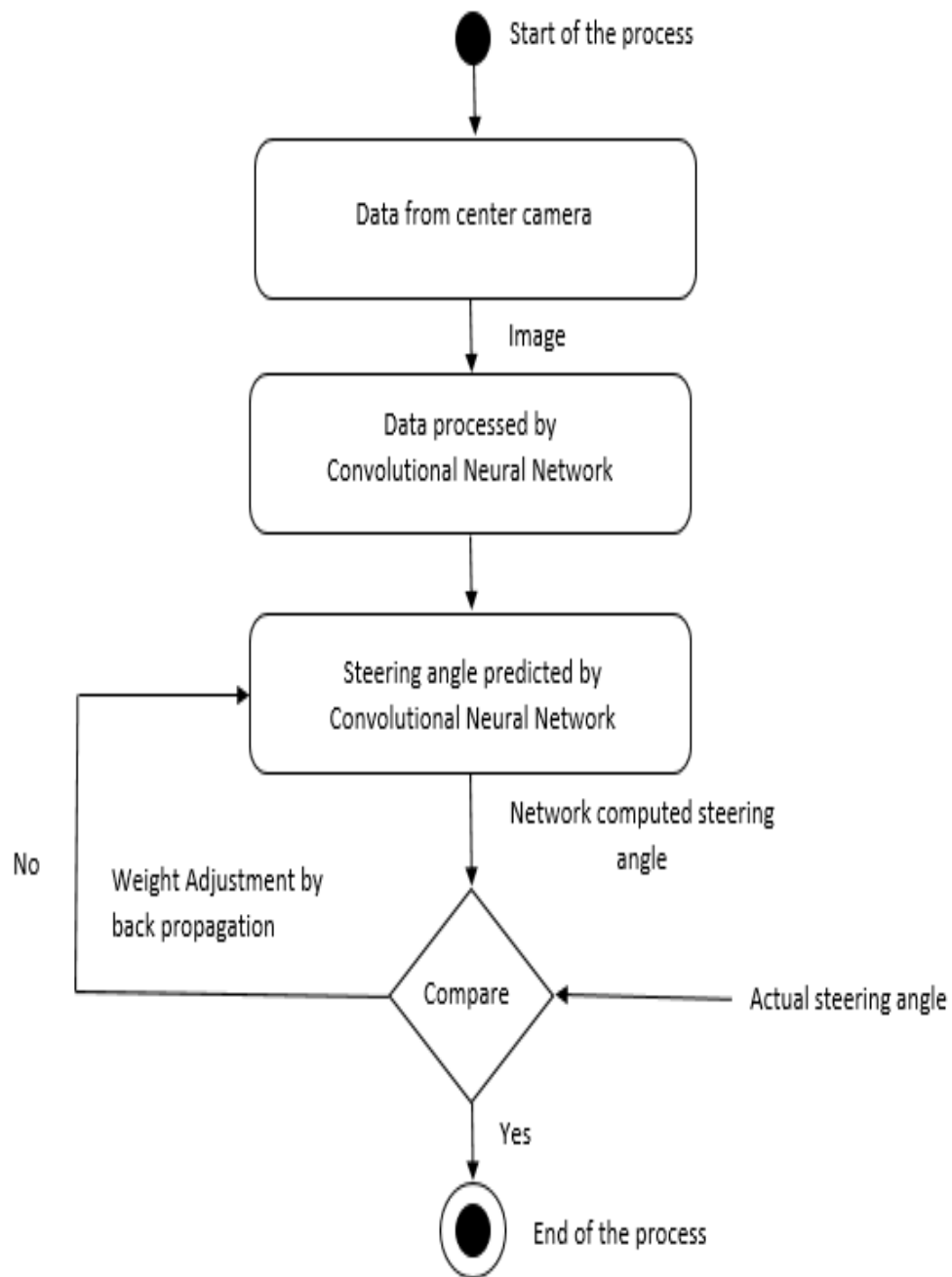


Fig 10. Activity Diagram 1

3.2 Activity Diagram for Testing Model:

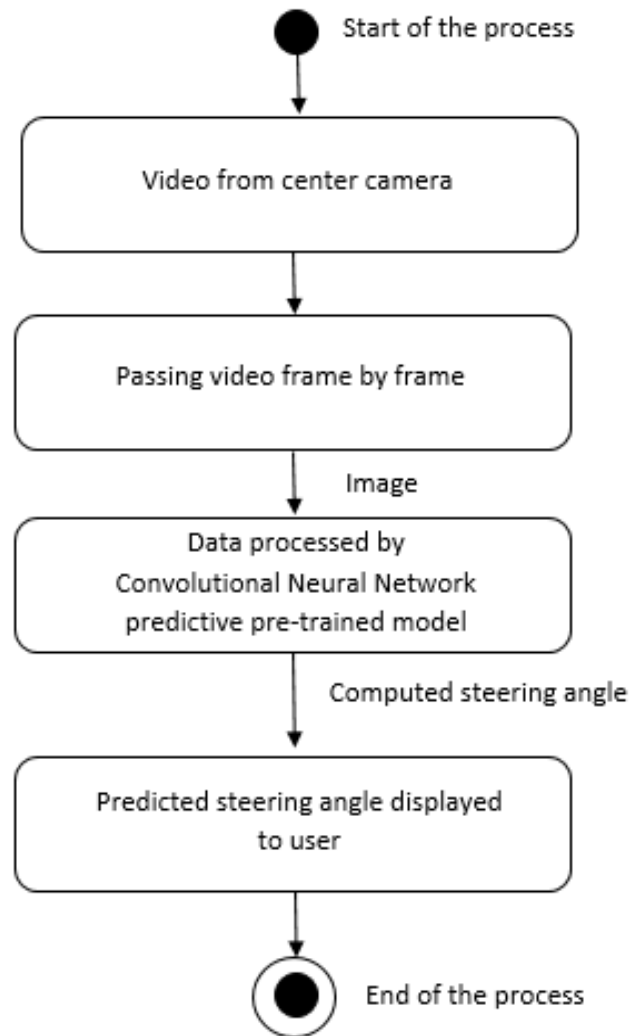


Fig 11. Activity Diagram 2

Implementation

To build the CNN model:

def build_cnn(image_size=None, weights_path=None):

This function build cnn model, which has 7 convolution layers, and 4 fully connected layers. The input to the function is image_size and weights_path. The function returns cnn model.

- The each layer of CNN is described below

x = Convolution2D(32, 3, 3, activation='relu', border_mode='same')(img_input)

In the first convolution layer here, we call the Convolution2D function with 32 conv filters each of size is 3*3. Here we use relu activation function. In addition, we use parameter border_mode is same which produce output equal to the input size. It performs conv operation on input image.

x = MaxPooling2D((2, 2), strides=(2, 2))(x)

In the MaxPooling2D function we perform maxpool operation on the output generate by previous Convolution2D function. Here we use 2*2 filter with stride is 2 which perform maxpooling operation.

x = Dropout(0.25)(x)

In the Dropout function we dropout 25% features generated by previous MaxPooling2D function to avoid overfitting.

x = Convolution2D(64, 3, 3, activation='relu', border_mode='same')(x)

In the second convolution layer here, we call the Convolution2D function with 64 conv filters each of size is 3*3. Here we use relu activation function. In addition, we use parameter border_mode is same which produce output equal to the input size. It performs conv operation on input image.

x = MaxPooling2D((2, 2), strides=(2, 2))(x)

In the MaxPooling2D function we perform maxpool operation on the output generate by previous Convolution2D function. Here we use 2*2 filter with stride is 2 which perform maxpooling operation.

x = Dropout(0.25)(x)

In the Dropout function we dropout 25% features generated by previous MaxPooling2D function to avoid overfitting.

x = Convolution2D(128, 3, 3, activation='relu', border_mode='same')(x)

In the third convolution layer here, we call the Convolution2D function with 128 conv filters each of size is 3*3. Here we use relu activation function. In addition, we use parameter border_mode is same which produce output equal to the input size. It performs conv operation on input image.

x = MaxPooling2D((2, 2), strides=(2, 2))(x)

Steering Angle Prediction for Self-Driving Car using Convolutional Neural Network

In the MaxPooling2D function we perform maxpool operation on the output generate by previous Convolution2D function. Here we use 2*2 filter with stride is 2 which perform maxpooling operation.

x = Dropout(0.25)(x)

In the Dropout function we dropout 25% features generated by previous MaxPooling2D function to avoid overfitting.

x = Convolution2D(256, 3, 3, activation='relu', border_mode='same')(x)

In the fourth convolution layer here, we call the Convolution2D function with 256 conv filters each of size is 3*3. Here we use relu activation function. In addition, we use parameter border_mode is same which produce output equal to the input size. It performs conv operation on input image.

x = MaxPooling2D((2, 2), strides=(2, 2))(x)

In the MaxPooling2D function we perform maxpool operation on the output generate by previous Convolution2D function. Here we use 2*2 filter with stride is 2 which perform maxpooling operation.

x = Dropout(0.5)(x)

In the Dropout function we dropout 25% features generated by previous MaxPooling2D function to avoid overfitting.

x = Convolution2D(256, 3, 3, activation='relu', border_mode='same')(x)

In the fifth convolution layer here, we call the Convolution2D function with 256 conv filters each of size is 3*3. Here we use relu activation function. In addition, we use parameter border_mode is same which produce output equal to the input size. It performs conv operation on input image.

x = MaxPooling2D((2, 2), strides=(2, 2))(x)

In the MaxPooling2D function we perform maxpool operation on the output generate by previous Convolution2D function. Here we use 2*2 filter with stride is 2 which perform maxpooling operation.

x = Dropout(0.5)(x)

In the Dropout function we dropout 25% features generated by previous MaxPooling2D function to avoid overfitting.

x = Convolution2D(512, 3, 3, activation='relu', border_mode='same')(x)

In the sixth convolution layer here, we call the Convolution2D function with 512 conv filters each of size is 3*3. Here we use relu activation function. In addition, we use parameter border_mode is same which produce output equal to the input size. It performs conv operation on input image.

x = MaxPooling2D((2, 2), strides=(2, 2))(x)

Steering Angle Prediction for Self-Driving Car using Convolutional Neural Network

In the MaxPooling2D function we perform maxpool operation on the output generate by previous Convolution2D function. Here we use 2*2 filter with stride is 2 which perform maxpooling operation.

x = Dropout(0.5)(x)

In the Dropout function we dropout 25% features generated by previous MaxPooling2D function to avoid overfitting.

x = Convolution2D(512, 3, 3, activation='relu', border_mode='same')(x)

In the seventh convolution layer here, we call the Convolution2D function with 512 conv filters each of size is 3*3. Here we use relu activation function. In addition, we use parameter border_mode is same which produce output equal to the input size. It performs conv operation on input image.

x = MaxPooling2D((2, 2), strides=(2, 2))(x)

In the MaxPooling2D function we perform maxpool operation on the output generate by previous Convolution2D function. Here we use 2*2 filter with stride is 2 which perform maxpooling operation.

x = Dropout(0.5)(x)

In the Dropout function we dropout 25% features generated by previous MaxPooling2D function to avoid overfitting.

- **Fully connected layer 4**

y = Flatten()(x)

In Flatten function, we flatten all the features extracted by convolutional layer in one vector.

y = Dense(4096, activation='relu')(y)

In Dense function, we create the fully connected layer using 4096 neurons and here we use relu activation function.

y = Dropout(.75)(y)

In the Dropout function we dropout 75% features generated by previous MaxPooling2D function to avoid overfitting.

y = Dense(2048, activation='relu')(y)

In Dense function, we create the fully connected layer using 2048 neurons and here we use relu activation function.

y = Dropout(.75)(y)

In the Dropout function we dropout 75% features generated by previous MaxPooling2D function to avoid overfitting.

```
y = Dense(1024, activation='relu')(y)
```

In Dense function, we create the fully connected layer using 1024 neurons and here we use relu activation function.

```
y = Dropout(.75)(y)
```

In the Dropout function we dropout 75% features generated by previous MaxPooling2D function to avoid overfitting.

```
y = Dense(512, activation='relu')(y)
```

In Dense function, we create the fully connected layer using 4096 neurons and here we use relu activation function.

```
y = Dropout(.75)(y)
```

In the Dropout function we dropout 75% features generated by previous MaxPooling2D function to avoid overfitting.

```
y = Dense(1)(y)
```

This dense layer generate output of our model.

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

Here we use Adam optimizer to train our model, we use the learning rate as 0.00001, and we use mean square error loss.

```
yhat = model.predict(test_x)
```

After training of model, we predict steering angle for our test dataset using model.predict function. This function returns predicted steering angle and we store it in to yhat.

- **For estimating accuracy of our model:**

Here we use root mean square method

```
for j in range(test_y.shape[0]):
```

```
    sqd = ((yhat[j]-test_y[j])**2)
```

```
    sq = sq + sqd
```

```
mse = sq/1600
```

```
rmse = np.sqrt(mse)
```

The above segment of code calculate root mean square error with respect to test dataset. Here yhat represent the predicted steering angle and test_y represent accurate steering angle.

- **To visualize the result:**

```
def draw_path_on(img, speed_ms, angle_steers, color=(0,0,255)):
```

Here we call function `draw_path_on` to draw the steering angle path on the image. The input parameter for this function is the input image, speed, predicted steering angle, and color of the path.

```
path_x = np.arange(0.0, 50.1, 0.5)
```

Here we arrange the 100 points using `np.arange` function. Here we need to calculate the look ahead offset of each point for that purpose we call following `calc_lookahead_offset` function

```
def calc_lookahead_offset(v_ego, angle_steers, d_lookahead, angle_offset=0):
```

The input parameters of function is speed, steering angle and lookahead distance. here we compute the curvature with respect to lookahead distance. to compute curvature we call `calc_curvature` function.

```
y_actual = d_lookahead * np.tan(np.arcsin(np.clip(d_lookahead * curvature, -0.999, 0.999))/2.)
```

After getting curvature, we calculate actual y valued with respect to look ahead distance by using above formula.

```
def calc_curvature(v_ego, angle_steers, angle_offset=0):
```

This function returns the lateral offset given the steering angle, speed and the look ahead distance. Here we use `slip_factor = 0.0014`, `steer_ratio = 15.3` and `wheel_base = 2.67` which is obtain from <http://www.edmunds.com/acura/ilx/2016/sedan/features-specs/> this site. Then we calculate the steer angle in radian by using following formula.

```
angle_steers_rad = (angle_steers - angle_offset) * deg_to_rad
```

After that, we compute the curvature by using following formula:

```
curvature = angle_steers_rad/(steer_ratio * wheel_base * (1. + slip_factor * v_ego**2))
```

In addition, return the calculated curvature.

```
def draw_path(img, path_x, path_y, color):
```

The input parameter of that function is input image `path_x`, which consist the look ahead distance and `path_y`, which consist the curvature with respect to that look ahead distance and color. In that, function we zip all the items from `path_x` and `path_y` and it gives to the following `draw_pt` function.

```
def draw_pt(img, x, y, color, sz=1):
```

The input parameter of function is input image, x, y, color and size which is defaulted to 1. This function compute the perspective transformation with respect to the x and y value.

```
img[row-sz:row+sz, col-sz:col+sz] = color
```

this line of code fill color in that point on the image.

Integration and Testing

Testing Performed:

Test Plan:

a. Time:

This project should predict steering angle as fast as possible and should immediately visualize the result. There are different methods implemented in this project, time taken by these methods are as follows:

Methods	Time Required
System Booting	12 seconds
Steering angle prediction	0.25 second
Visualizing the result	10 frames in 1 seconds

b. Output:

1. Accurate steering angle:

As we are detecting steering angle based on previous results it is necessary to pass the previous results while training the model. We observe that this method gives results that are more accurate.

Performance Analysis

Performance Requirement:

The following table shows the performance of each method:

Methods	Time Required
System Booting	12 seconds
Steering angle prediction	0.25 second
Visualizing the result	10 frames in 1 seconds

We run above methods on the CPU. The system will take much time to boot on CPU. It is required that the system should be booted in 1 sec.

After running the above methods on the GPU, it satisfies the performance requirement. CUDA enables increase in computing performance by harnessing power of GPU. It processes more frame to give accurate results quickly.

Applications

Steering Angle Prediction for Self-Driving Car using Convolutional Neural Network

- It can be used in Advanced Driver Assistance System.
- It ensures safe driving.

Installation Guide and User Manual.

1. Installing Python 3.6:

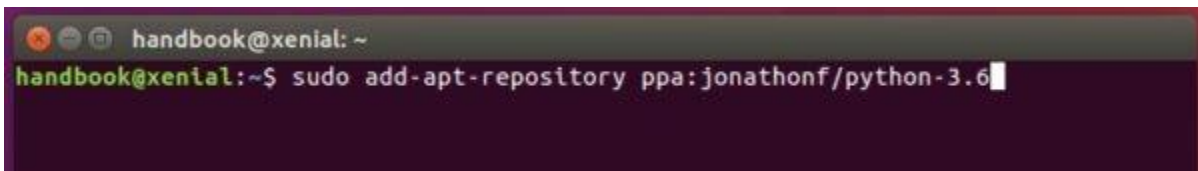
Python 3.6 is most compatible version.

Ubuntu 16.04 comes with both Python 2.7 and Python 3.5 by default. You can install Python 3.6 along with them via a [third party PPA](#) by doing following steps:

1. Open terminal via Ctrl+Alt+T or searching for “Terminal” from app launcher. When it opens, run command to add the PPA:

```
sudo add-apt-repository ppa:jonathonf/python-3.6
```

Type in your password (no visual feedback due to security reason) when it asks and hit Enter.

A terminal window with a dark background. The prompt is 'handbook@xenial: ~'. The command 'sudo add-apt-repository ppa:jonathonf/python-3.6' is entered and the cursor is at the end of the line.

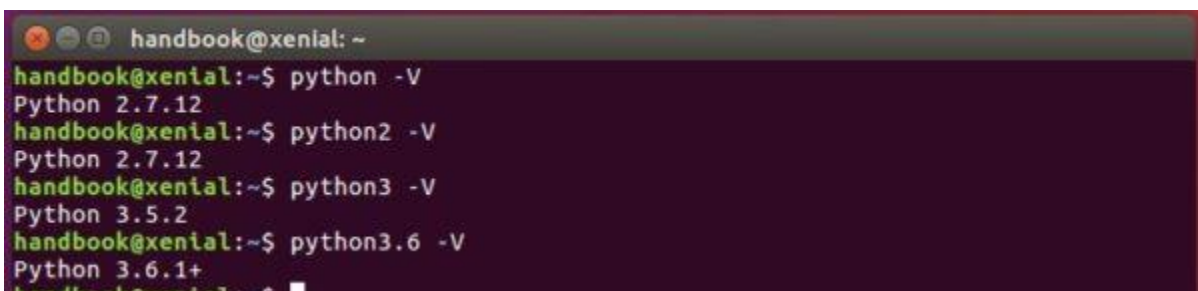
```
handbook@xenial: ~  
handbook@xenial:~$ sudo add-apt-repository ppa:jonathonf/python-3.6
```

2. Then check updates and install Python 3.6 via commands:

```
sudo apt-get update
```

```
sudo apt-get install python3.6
```

Now you have three Python versions, use `python` command for version 2.7, `python3` for version 3.5, and/or `python3.6` for version 3.6.1.

A terminal window with a dark background. The prompt is 'handbook@xenial: ~'. The following commands and their outputs are shown:

```
handbook@xenial:~$ python -V  
Python 2.7.12  
handbook@xenial:~$ python2 -V  
Python 2.7.12  
handbook@xenial:~$ python3 -V  
Python 3.5.2  
handbook@xenial:~$ python3.6 -V  
Python 3.6.1+  
handbook@xenial:~$
```

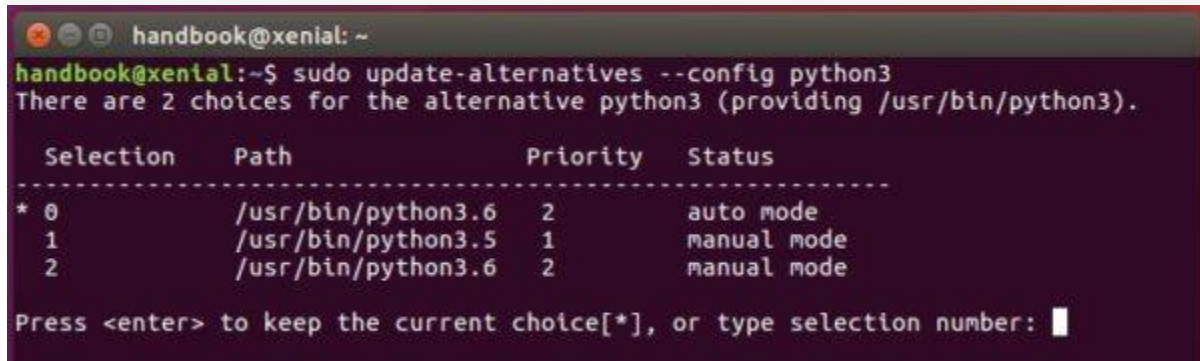
3. To make `python3` use the new installed python 3.6 instead of the default 3.5 release, run following 2 commands:

```
sudo update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.5 1
```

```
sudo update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.6 2
```

Finally switch between the two python versions for `python3` via command:

```
sudo update-alternatives --config python3
```



```
handbook@xenial: ~  
handbook@xenial:~$ sudo update-alternatives --config python3  
There are 2 choices for the alternative python3 (providing /usr/bin/python3).  
  
  Selection    Path                        Priority  Status  
-----  
*  0            /usr/bin/python3.6          2        auto mode  
    1            /usr/bin/python3.5          1        manual mode  
    2            /usr/bin/python3.6          2        manual mode  
  
Press <enter> to keep the current choice[*], or type selection number: 
```

After selecting version 3.6:

```
python3 -V
```



```
handbook@xenial: ~  
handbook@xenial:~$ python3 -V  
Python 3.6.1+  
handbook@xenial:~$ 
```

2. Installing TensorFlow:

Step 1 — Installing TensorFlow

In this step, we are going to create a virtual environment and install TensorFlow.

First, create a project directory called tf-demo:

```
mkdir ~/tf-demo
```

Navigate to your newly created tf-demo directory:

```
cd ~/tf-demo
```

Steering Angle Prediction for Self-Driving Car using Convolutional Neural Network

Then create a new virtual environment called `tensorflow-dev`. Run the following command to create the environment:

```
python3 -m venv tensorflow-dev
```

This creates a new tensorflow-dev directory, which will contain all of the packages that you install while this environment is activated. It also includes pip and a standalone version of Python.

Now activate your virtual environment:

```
source tensorflow-dev/bin/activate
```

Once activated, you will see something similar to this in your terminal:

```
(tensorflow-dev)username@hostname:~/tf-demo $
```

Now you can install TensorFlow in your virtual environment.

Run the following command to install and upgrade to the newest version of TensorFlow available in PyPi:

```
pip3 install --upgrade tensorflow
```

TensorFlow will install:

Output

Collecting tensorflow

Downloading tensorflow-1.4.0-cp36-cp36m-macosx_10_11_x86_64.whl (39.3MB)

100% 39.3MB 35kB/s

...

Successfully installed bleach-1.5.0 enum34-1.1.6 html5lib-0.9999999 markdown-2.6.9 numpy-1.13.3
protobuf-3.5.0.post1 setuptools-38.2.3 six-1.11.0 tensorflow-1.4.0 tensorflow-tensorboard-0.4.0rc3
werkzeug-0.12.2 wheel-0.30.0

Step 2 — Validating Installation

To validate the installation of TensorFlow, we are going to run a simple program in TensorFlow as a non-root user. We will use the canonical beginner's example of "Hello, world!" as a form of validation. Rather than creating a Python file, we'll create this program using Python's interactive console.

To write the program, start up your Python interpreter:

```
python
```

You will see the following prompt appear in your terminal

```
>>>
```

This is the prompt for the Python interpreter, and it indicates that it's ready for you to start entering some Python statements.

First, type this line to import the TensorFlow package and make it available as the local variable `tf`. Press ENTER after typing in the line of code:

```
import tensorflow as tf
```

Next, add this line of code to set the message "Hello, world!":

```
hello = tf.constant("Hello, world!")
```

Then create a new TensorFlow session and assign it to the variable `sess`:

```
sess = tf.Session()
```

Note: Depending on your environment, you might see this output:

Finally, enter this line of code to print out the result of running the hello TensorFlow session you've constructed in your previous lines of code:

```
print(sess.run(hello))
```

Installing Anaconda

The best way to install Anaconda is to download the latest Anaconda installer bash script, verify it, and then run it.

Find the latest version of Anaconda for Python 3 at the [Anaconda Downloads page](#). At the time of writing, the latest version is 5.0.1, but you should use a later stable version if it is available.

Next, change to the /tmp directory on your server. This is a good directory to download ephemeral items, like the Anaconda bash script, which we won't need after running it.

```
cd /tmp
```

Use curl to download the link that you copied from the Anaconda website:

```
curl -O https://repo.continuum.io/archive/Anaconda3-5.0.1-Linux-x86_64.sh
```

We can now verify the data integrity of the installer with cryptographic hash verification through the SHA-256 checksum. We'll use the sha256sum command along with the filename of the script:

```
sha256sum Anaconda3-5.0.1-Linux-x86_64.sh
```

You'll receive output that looks similar to this:

Output

```
55e4db1919f49c92d5abfb27a4be5986ae157f074bf9f8238963cd4582a4068a  Anaconda3-5.0.1-Linux-x86_64.sh
```

You should check the output against the hashes available at the [Anaconda with Python 3 on 64-bit Linux page](#) for your appropriate Anaconda version. As long as your output matches the hash displayed in the sha256 row then you're good to go.

Now we can run the script:

```
bash Anaconda3-5.0.1-Linux-x86_64.sh
```

Output

```
Welcome to Anaconda3 5.0.1 (by Continuum Analytics, Inc.)
```

In order to continue the installation process, please review the license agreement.

Please, press ENTER to continue

Press ENTER to continue and then press ENTER to read through the license. Once you're done reading the license, you'll be prompted to approve the license terms:

Output

Do you approve the license terms? [yes|no]

As long as you agree, type yes.

At this point, you'll be prompted to choose the location of the installation. You can press ENTER to accept the default location, or specify a different location to modify it.

Output

Anaconda3 will now be installed into this location:

/home/**sammy**/anaconda3

- Press ENTER to confirm the location
- Press CTRL-C to abort the installation
- Or specify a different location below

[/home/**sammy**/anaconda3] >>>

The installation process will continue, it may take some time.

Once it's complete you'll receive the following output:

Output

...

installation finished.

Do you wish the installer to prepend the Anaconda3 install location to PATH in your /home/sammy/.bashrc ? [yes|no]

[no] >>>

Type yes so that you can use the conda command. You'll next see the following output:

Output

Prepending PATH=/home/sammy/anaconda3/bin to PATH in /home/sammy/.bashrc

A backup will be made to: /home/sammy/.bashrc-anaconda3.bak

...

In order to activate the installation, you should source the ~/.bashrc file:

```
source ~/.bashrc
```

Once you have done that, you can verify your install by making use of the conda command, for example with list:

You'll receive output of all the packages you have available through the Anaconda installation:

Output

```
# packages in environment at /home/sammy/anaconda3:
```

```
#
```

```
_ipyw_jlab_nb_ext_conf 0.1.0      py36he11e457_0
```

```
alabaster              0.7.10     py36h306e16b_0
```

```
anaconda               5.0.1      py36hd30a520_1
```

```
...
```

Now that Anaconda is installed, we can go on to setting up Anaconda environments.

Setting Up Anaconda Environments

Anaconda virtual environments allow you to keep projects organized by Python versions and packages needed. For each Anaconda environment you set up, you can specify which version of Python to use and can keep all of your related programming files together within that directory.

First, we can check to see which versions of Python are available for us to use:

```
conda search "^python$"
```

You'll receive output with the different versions of Python that you can target, including both Python 3 and Python 2 versions. Since we are using the Anaconda with Python 3 in this tutorial, you will have access only to the Python 3 versions of packages.

Let's create an environment using the most recent version of Python 3. We can achieve this by assigning version 3 to the python argument. We'll call the environment **my_env**, but you'll likely want to use a more descriptive name for your environment especially if you are using environments to access more than one version of Python.

```
conda create --name my_env python=3
```

We'll receive output with information about what is downloaded and which packages will be installed, and then be prompted to proceed with y or n. As long as you agree, type y.

The conda utility will now fetch the packages for the environment and let you know when it's complete.

You can activate your new environment by typing the following:

```
source activate my_env
```

With your environment activated, your command prompt prefix will change:

Steering Angle Prediction for Self-Driving Car using Convolutional Neural Network

Within the environment, you can verify that you're using the version of Python that you had intended to use:

```
python --version
```

Output

Python 3.6.0 :: Continuum Analytics, Inc.

When you're ready to deactivate your Anaconda environment, you can do so by typing:

```
source deactivate
```

Note that you can replace the word source with . to achieve the same results.

To target a more specific version of Python, you can pass a specific version to the python argument, like 3.5, for example:

```
conda create -n my_env35 python=3.5
```

You can update your version of Python along the same branch (as in updating Python 3.5.1 to Python 3.5.2) within a respective environment with the following command:

```
conda update python
```

If you would like to target a more specific version of Python, you can pass that to the python argument, as in python=3.3.2.

You can inspect all of the environments you have set up with this command:

```
conda info --envs
```

Output

```
# conda environments:
```

```
#
```

```
my_env          /home/sammy/anaconda3/envs/my_env
```

```
my_env35        /home/sammy/anaconda3/envs/my_env35
```

```
root            * /home/sammy/anaconda3
```

The asterisk indicates the current active environment.

Each environment you create with conda create will come with several default packages:

- openssl

Steering Angle Prediction for Self-Driving Car using Convolutional Neural Network

- pip
- python
- readline
- setuptools
- sqlite
- tk
- wheel
- xz
- zlib

You can add additional packages, such as numpy for example, with the following command:

```
conda install --name my_env35 numpy
```

If you know you would like a numpy environment upon creation, you can target it in your conda createcommand:

```
conda create --name my_env python=3 numpy
```

If you are no longer working on a specific project and have no further need for the associated environment, you can remove it. To do so, type the following:

```
conda remove --name my_env35 --all
```

Now, when you type the conda info --envs command, the environment that you removed will no longer be listed.

Updating Anaconda

You should regularly ensure that Anaconda is up-to-date so that you are working with all the latest package releases.

To do this, you should first update the conda utility:

```
conda update conda
```

When prompted to do so, type y to proceed with the update.

Once the update of conda is complete, you can update the Anaconda distribution:

```
conda update anaconda
```

Steering Angle Prediction for Self-Driving Car using Convolutional Neural Network
Again when prompted to do so, type y to proceed.

This will ensure that you are using the latest releases of conda and Anaconda.

3. Installing Keras:

Install Keras from PyPI (recommended):

```
sudo pip install keras
```

If you are using a virtualenv, you may want to avoid using sudo:

```
pip install keras
```

Alternatively: install Keras from the GitHub source:

First, clone Keras using git:

```
git clone https://github.com/keras-team/keras.git
```

Then, cd to the Keras folder and run the install command:

```
cd keras  
sudo python setup.py install
```

4. Installing CUDA:

1. Update the system

```
apt-get update && apt-get upgrade
```

2. Download VirtualGL and install it. To install

```
dpkg -i virtualgl*.deb
```

3. Download and install CUDA 8.0 and install it. I suggest to do it vs through the internet. As like this,

For Linux users upgrading from previous versions of the CUDA Toolkit, this page may assist in the system setup process.

For developers on x86-64 systems using Tesla P100 GPUs, click to see instructions before installation..

Select Target Platform ⓘ		Related Links
Click on the green buttons that describe your target platform. Only supported platforms will be shown.		CUDA Quick Start Guide Release Notes EULA Online Documentation CUDA Toolkit Overview Installer Checksums Open Source Packages Legacy CUDA Toolkits
Operating System	Windows Linux Mac OSX	
Architecture ⓘ	x86_64 ppc64le	
Distribution	Fedora OpenSUSE RHEL CentOS SLES Ubuntu	
Version	16.04 14.04	
Installer Type ⓘ	runfile (local) deb (local) deb (network) cluster (local)	

Download Installer for Linux Ubuntu 16.04 x86_64

The base installer is available for download below.

[Base Installer](#) [Download \[2.6 KB\]](#)

Installation Instructions:

```
1. "sudo dpkg -i cuda-repo-ubuntu1604_8.0.44-1_amd64.deb"  
2. "sudo apt-get update"  
3. "sudo apt-get install cuda"
```

The CUDA Toolkit contains Open-Source Software. The source code can be found [here](#).
The checksums for the installer and patches can be found in [Installer Checksums](#).
For further information, see the [Installation Guide for Linux](#) and the [CUDA Quick Start Guide](#).

4. Install required dependencies.

```
apt-get install linux-headers-$(uname -r)
apt-get install freeglut3-dev libxmu-dev libpcap-dev
```

5. Update system PATH in .bashrc which can be found in the home directory. Please note if you install those thing into difference location, please update path according to that.

```
export PATH=$PATH:/opt/VirtualGL/bin
export PATH=$PATH:/usr/local/cuda/bin
```

6. Install bumblebee-nvidia and primus.
apt-get install bumblebee-nvidia primus

7. Edit the bumblebee config file so bumblebee knows we are using the NVIDIA driver. Please update the path according to your system. Here is reference view which will help.

```
sudo nano +22 /etc/bumblebee/bumblebee.conf
```

Add:

```
[bumblebeed]
ServerGroup=bumblebee
TurnCardOffAtExit=false
NoEcoModeOverride=false
Driver=nvidia
XorgConfDir=/etc/bumblebee/xorg.conf.d
Bridge=auto
PrimusLibraryPath=/usr/lib/x86_64-linux-gnu/primus:/usr/lib/i386-linux-gnu/primus
AllowFallbackToIGC=false
Driver=nvidia
[driver-nvidia]
KernelDriver=nvidia
PMMethod=auto
LibraryPath=/usr/lib/nvidia-367:/usr/lib32/nvidia-367
XorgModulePath=/usr/lib/xorg,/usr/lib/xorg/modules
XorgConfFile=/etc/bumblebee/xorg.conf.nvidia
Driver=nouveau
[driver-nouveau]
KernelDriver=nouveau
PMMethod=auto
XorgConfFile=/etc/bumblebee/xorg.conf.nouveau
```

8. Run the following and record the PCI address of your video card.
\$ lspci | egrep 'VGA|3D'
00:02.0 VGA compatible controller: Intel Corporation Device 5916 (rev 02)
01:00.0 3D controller: NVIDIA Corporation Device 179c (rev a2)

Steering Angle Prediction for Self-Driving Car using Convolutional Neural Network

9. Edit the `xorg.conf.nvidia` file so it knows the PCI address(01:00.0 for me) of your video card. Update PIC address as below under section "ServerLayout"

```
sudo nano /etc/bumblebee/xorg.conf.nvidia
```

Add:

Section "ServerLayout"

Identifier "Layout0"

Option "AutoAddDevices" "false"

Option "AutoAddGPU" "false"

BusID "PCI:01:00.0"

10. Reboot the system and have a fun with running some sample codes.

```
sudo shutdown -r now
```

Ethics

Declaration of Ethics:

As a computer Science & Engineering Student, I believe it is Unethical To,

- i. Surf the internet for personal interest and non-class related purposes during classes.
- ii. Make a copy of software for personal or commercial use.
- iii. Make a copy of software for friend.
- iv. Loan CDs of Software to friends.
- v. Download pirated software from the internet.
- vi. Distribute pirated software from the internet.
- vii. Buy software with a single user license and then install it on multiple Computers.
- viii. Share a pirated copy of software.
- ix. Install a pirated copy of software.

References

Steering Angle Prediction for Self-Driving Car using Convolutional Neural Network

- Nvidia : End to End Learning for Self Driving Cars by Mariusz Bojarski, Beat Flepp, Urs Muller, Davide Del Testa, Prasoon Goyal, Jiakai Zhang, Daniel Dworakowski, Lawrence D. Jackel, Xin Zhang, Bernhard Firner, Mathew Monfort, Jake Zhao, Karol Zieba
- Deep learning for Video classification and captioning by Zuxuan Wu, Ting Yao, Yanwei Fu, Yu-Gang Jiang
- Visualizing and Understanding Convolutional Networks by Matthew D. Zeiler, Rob Fergus