VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SELF DRIVING CAR IN A SIMULATED ENVIRONMENT USING DEEP CONVOLUTIONAL NEURAL NETWORKS

An J Component Report

submitted by

TEAM MEMBERS:

NAME	REG. NO.
S.MAHAAN MALESH	17BCB0046

in partial fulfillment for the award of the degree of

B.Tech

in

COMPUTER SCIENCE ENGINEERING

Under the guidence

of

Faculty: Prof. Delhi Babu R

School of Computing Science And Engineering

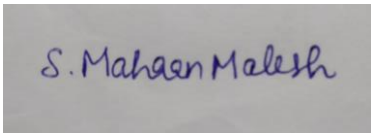
APRIL 2019

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

DECLARATION

I hereby declare that the J Component report entitled"**SELF DRIVING CAR IN A SIMULATED ENVIRONMENT USING DEEP CONVOLUTIONAL NEURAL NETWORKS"** submitted by me to Vellore Institute of Technology, Vellore-14 in partial fulfillment of the requirement for the award of the degree of **B.Tech** in **Computer science and engineering** is a record of bonafide undertaken by me under the supervision of **Dr. R. Delhi Babu** I further declare that the work reported in this report has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.



Signature
Name: S.MAHAAN MALESH
Reg. Number: 17BCB0046

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

The aim of the project is to create an AI that automatically navigates through a game environment . In this project the environment is the game NFS(Need For Speed Most Wanted) and the agent is any car that can navigate through the environment. I have used AlexNet , a Deep Convolutional Neural Network. The training data for the convolutional neural network is a one hour of gameplay video created by myself for the project. The training data collected is released publicly under cc license.

1.1 Abstract

The aim of this project is to understand and create a self driving vehicle in a simulated environment. The simulated environment can be a game, sandbox environment or virtual environment specifically designed for testing physics of cars. Image from the virtual environment is stored as numpy array and deep neural network is applied on the image for further processing. Various techniques will be used to derive information from the region of interest in the image and then the image data is used to train a convolutional neural network. Convolutional neural networks are deep artificial neural networks that are used primarily to classify images (e.g. name what they see), cluster them by similarity (photo search), and perform object recognition within scenes. They are algorithms that can identify faces, individuals, street signs and many other aspects of visual data. Eventually the model can be deployed for real world applications once the accuracy is met in the simulated environment.

Training data consist of the screenshot of the game with the corresponding key presses. One hot encoding is used to represent key presses. Then the image data and the key press data is

given as input to the Alexnet. The alexnet in turn generates the probability of each key presses. The key press with the highest probability is then selected to maneuver.

1.2 REQUIREMENTS

Software specification:

- Compiler must support tensor flow environment
- Python 3.6.0

  Windows 7 or above or any linux distribution.

hardware specification

- 8 GB RAM
- 20 GB Disk Space
- GPU is a plus!

libraries

- NUMPY
- PYAUTOGUI
- MATPLOTLIB
- SCIPY
- ITERTOOLS
- SKLEARN
- KERAS
- TENSORFLOW
- OPENCV

1.3 INPUT

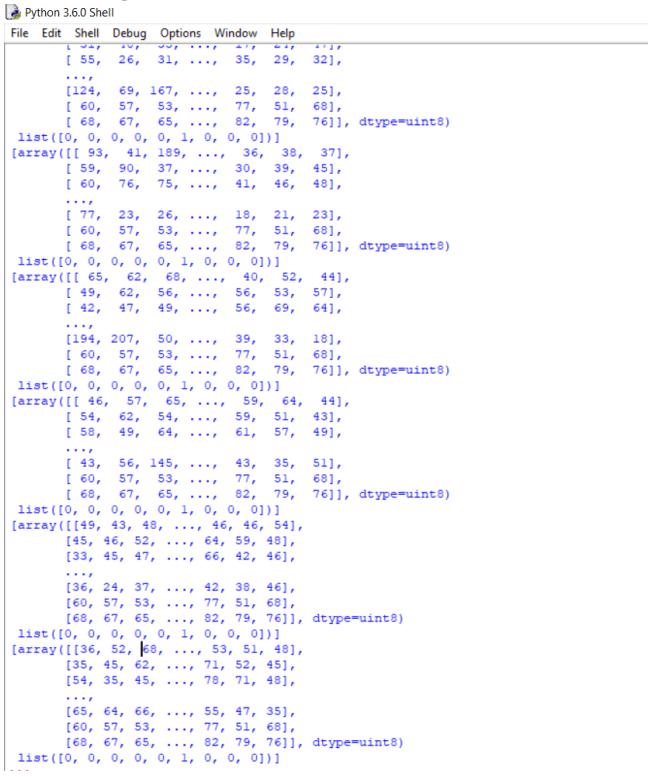
Figure:1 Input

Figure:2 Input

Keypresses are represented in one hot encoding format:
w = [1,0,0,0,0,0,0,0,0]
s = [0,1,0,0,0,0,0,0,0]
a = [0,0,1,0,0,0,0,0,0]
d = [0,0,0,1,0,0,0,0,0]
wa = [0,0,0,0,1,0,0,0,0]
wd = [0,0,0,0,0,1,0,0,0]
sa = [0,0,0,0,0,0,1,0,0]
sd = [0,0,0,0,0,0,0,1,0]
nk = [0,0,0,0,0,0,0,0,1]

To reduce the load on neural network the 800*600 RGB image from the game is resized to 160*120 RGB image

Training data:



1.4 OUTPUT

Outputs:

Figure 3:output

The time taken to process each frame and the probability of each keypress is displayed.
The key with the highest probability is pressed by the AI.

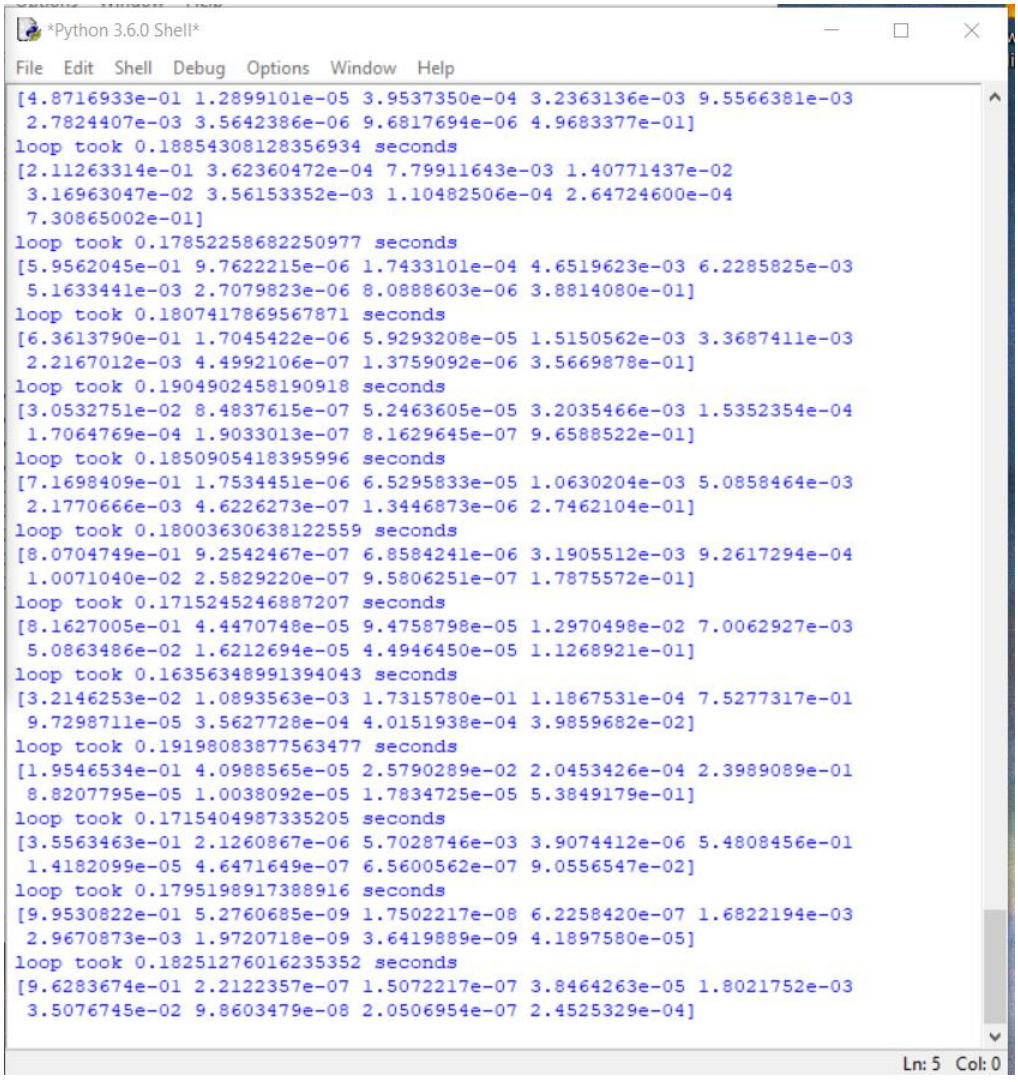


Figure 4:output

1.5 SAMPLE CODE USED TO GENERATE TRAINING DATA

```
def keys_to_output(keys):
    '''
    Convert keys to a ...multi-hot... array
     0  1  2  3  4   5   6   7    8
    [W, S, A, D, WA, WD, SA, SD, NOKEY] boolean values.
    '''
    output = [0,0,0,0,0,0,0,0,0]

    if 'W' in keys and 'A' in keys:
        output = wa
    elif 'W' in keys and 'D' in keys:
```

```
        output = wd
    elif 'S' in keys and 'A' in keys:
        output = sa
    elif 'S' in keys and 'D' in keys:
        output = sd
    elif 'W' in keys:
        output = w
    elif 'S' in keys:
        output = s
    elif 'A' in keys:
        output = a
    elif 'D' in keys:
        output = d
    else:
        output = nk
    return output

while(True):
    if not paused:
        # 800x600 windowed mode
        screen = grab_screen(region=(0,40,800,640))
        last_time = time.time()
        screen = cv2.cvtColor(screen, cv2.COLOR_BGR2RGB)
        screen = cv2.resize(screen, (160,120))
        keys = key_check()
        output = keys_to_output(keys)
        training_data.append([screen,output])

        if len(training_data) % 1000 == 0:
            print(len(training_data))
            np.save(file_name,training_data)

    keys = key_check()
```

2. DEEP NEURAL NETWORK IMPLEMENTATION

2.1 ALEXNET EXPLANATION:

In this project I'm using AlexNet a deep convolutional neural network which learns visual knowledge from images to classify a given image. It had produced breakthrough results in ImageNet LSVRC-2010 contest achieving a top-1 error rate of 37.5 % which was better than previous start of art methods which achieved 47.1%. The network consists of 8 layers out of which 5 were convolutional layers and other 3 were fully connected layers. The below Section details its architecture.

Layer 1

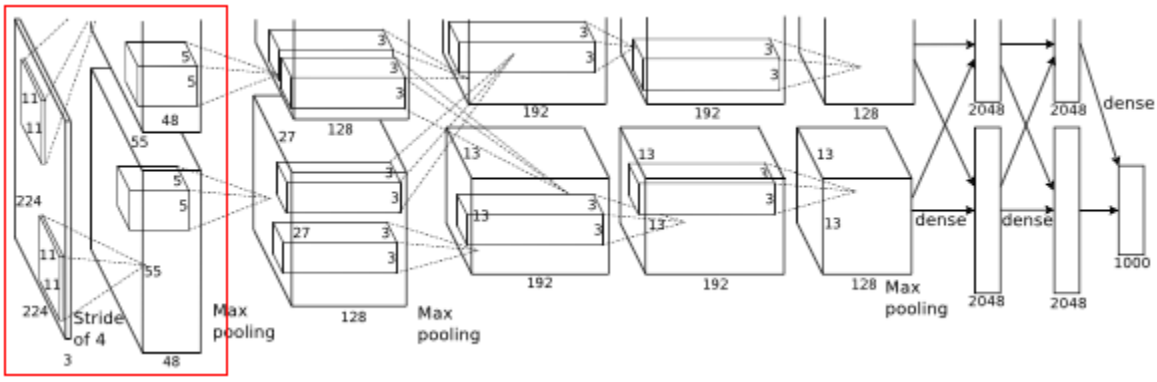


Figure 5:Alexnet

Before image(s) is given to this layer, variable resolution images in training set are re-scaled to a fixed size of 256 * 256 as deep neural networks expects all inputs to be of fixed size. This is done by first rescaling the image such that shorter side is of length 256, and then central 256*256 is cropped out.

To prevent overfitting the training images on this net, dropout of 0.5 in some layers and two methods of data augmentation where labels are preserved were used. One is in which random 224 * 224 patches(and their horizontal reflections) were extracted from each image. Thus this method increase the size of training set by reasonable amount. Another is to perform a transformation to vary the intensity and color of illumination for a image as object identity is invariant to these images.

And Rectified Linear unit neurons are used, where its activation function is max(0,x). By using these neurons, training time will be decreased according to the observations.

- Then this 224 * 224 * 3 image is given to 1st convolutional layer.
- Where this layer filters the image with 96 kernels of size 11 * 11 * 3
- Stride - 4, assuming padding is 2 Here stride(s) < kernel size(z) (4 < 11), so this overlapping pooling. According to the paper, the models with overlapping pooling find it slightly more difficult to overfit.
- Therefore output = (224 - 11 + 2 *2)/4 + 1 = 55
- Output is of size 55 * 55 * 96
- To this output, local response normalization(LRN) is applied which is a brightness normalization.
- After LRN, max pooling layer is applied where stride is 2 and kernel size is 3 * 3
- Therefore now output is 27((55 - 3)/2 + 1) * 27 * 96

Layer 2

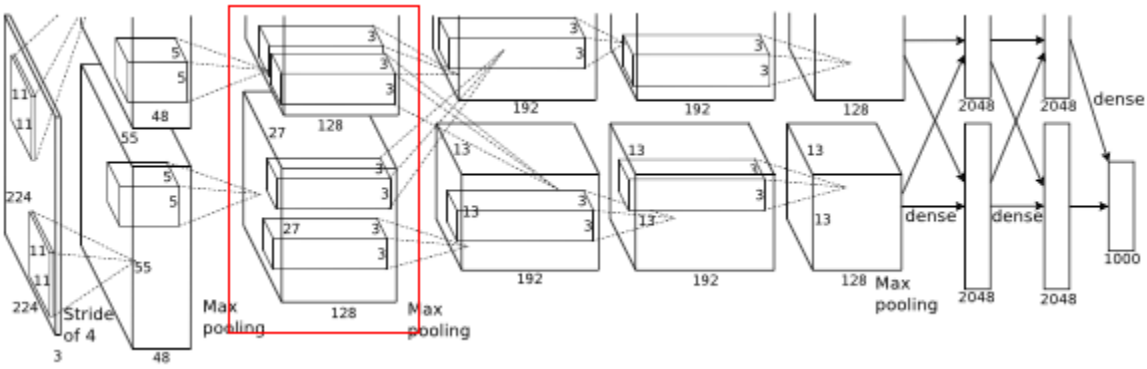


Figure 6:Alexnet

- Input to this layer is 27 * 27 * 96
- Filters = 256
- Kernel size = 5 * 5 * 48, padding = 2
- Stride = 1
- Output width or height = (27 - 5 + 2 * 2)/1 + 1 = 27
- Output is 27 * 27 * 256
- This is followed by LRN and max pooling layer
- Final output layer is
    - Max pooling kernel of size 3 * 3 and stride 1
    - Output width or height = (27 - 3 + 2)/1 + 1 = 27
    - Final output = 27 * 27 * 256

Layer 3

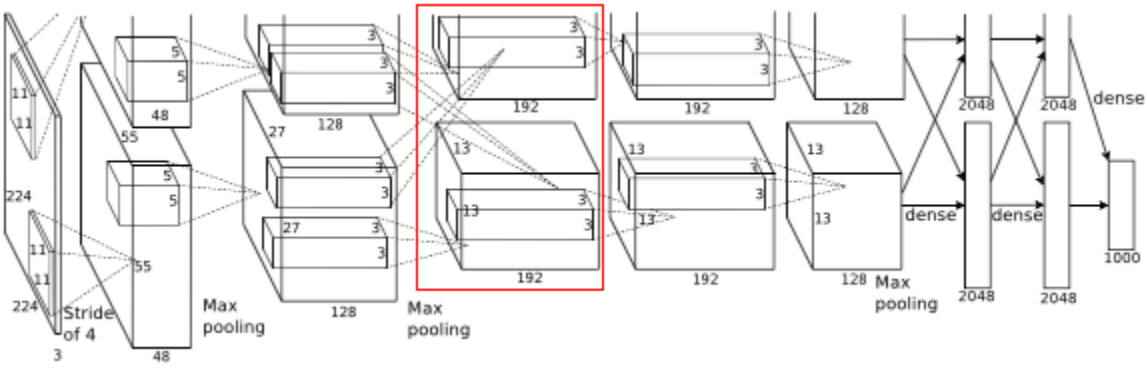


Figure 7:Alexnet

- Input to this layer is 27 * 27 * 256
- Filters = 384
- Kernel size 3 * 3 *256, padding - 1

- Output width or height = (27 - 3 + 2)/1 + 1 = 27
- Final output is 27 * 27 * 384

Layer 4

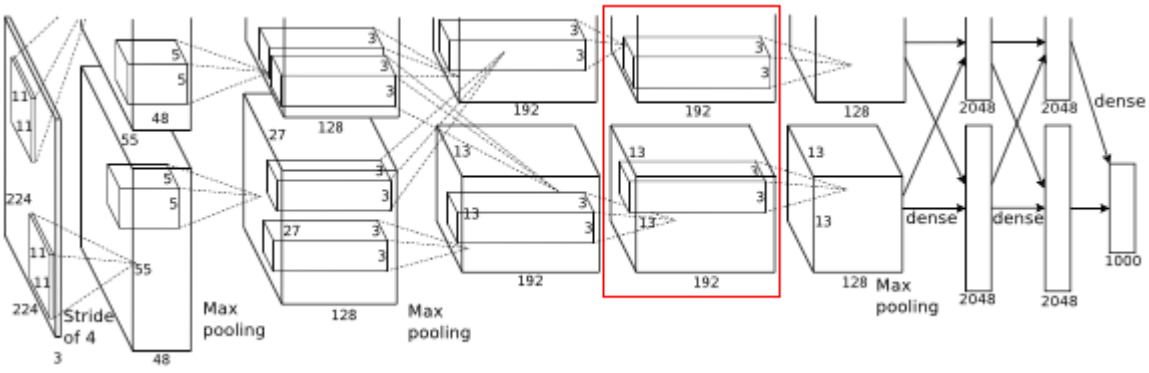


Figure 8:Alexnet

- Input to this layer is 27 * 27 * 384
- Filters = 384
- Kernel size is 3 * 3 * 192, padding = 1
- Output width or height = (27 - 3 + 2)/1 + 1 = 27
- Final output is 27 * 27 * 384

Layer 5

Similarly after the fifth convolutional layer

- As filters = 256
- Output is 27 * 27 * 256
- After the max pooling layer
    - Kernel size is 3 * 3 and stride 2
    - Final output width or height = (27 - 3)/2 + 1 = 13
    - Final output is 13 * 13 * 256 And other two layers are fully connected layers of 4096 each

Softmax

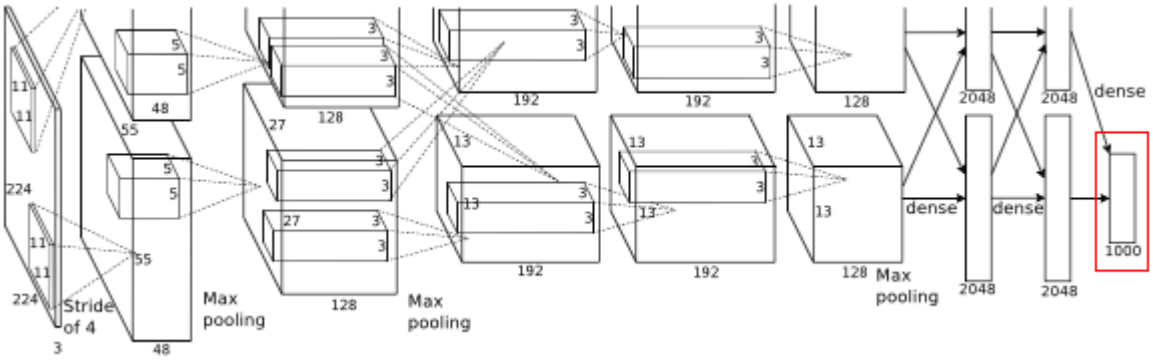


Figure 9:Alexnet

To the final layer 1000 way softmax is used to get predicted probabilities for each class.

The above alexnet is modified slightly i.e an input of 160*120 is given and the output is a 9 way softmax function is used to predict the probabilities for each class.

2.2 ALEXNET CODE:

```
def alexnet(width, height, lr, output=3):

    network = input_data(shape=[None, width, height, 1], name='input')

    network = conv_2d(network, 96, 11, strides=4, activation='relu')

    network = max_pool_2d(network, 3, strides=2)

    network = local_response_normalization(network)

    network = conv_2d(network, 256, 5, activation='relu')

    network = max_pool_2d(network, 3, strides=2)

    network = local_response_normalization(network)

    network = conv_2d(network, 384, 3, activation='relu')

    network = conv_2d(network, 384, 3, activation='relu')

    network = conv_2d(network, 256, 3, activation='relu')
```

```
network = max_pool_2d(network, 3, strides=2)

network = local_response_normalization(network)

network = fully_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully_connected(network, 4096, activation='tanh')

network = dropout(network, 0.5)

network = fully_connected(network, output, activation='softmax')

network = regression(network, optimizer='momentum',

                     loss='categorical_crossentropy',

                     learning_rate=lr, name='targets')


model = tflearn.DNN(network, checkpoint_path='model_alexnet',

                    max_checkpoints=1, tensorboard_verbose=2, tensorboard_dir='log')


return model
```

2.3SCREENSHOT

Screenshots:

During Training:

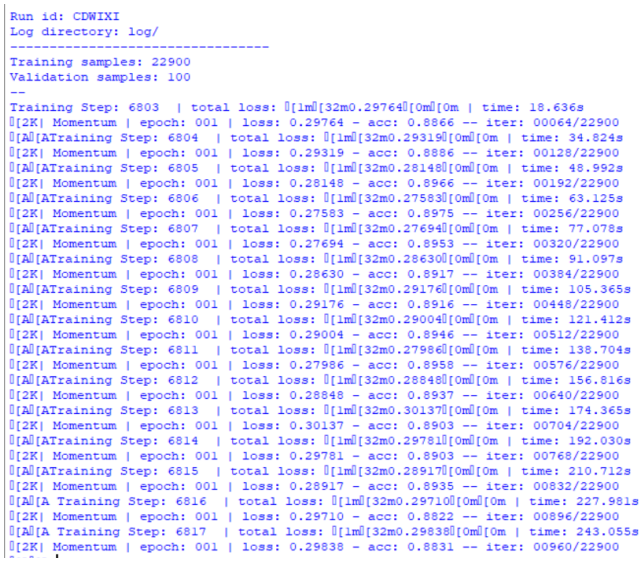


Figure 10

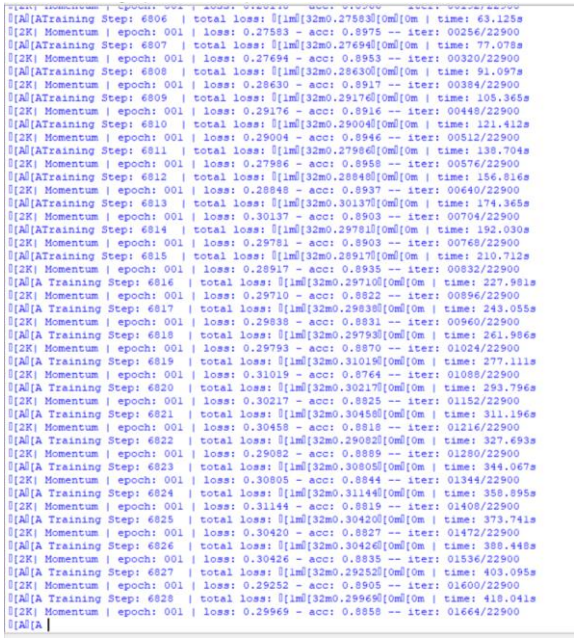


Figure 11

During Testing:

Figure 12


Python 3.6.0 Shell
File Edit Shell Debug Options Window Help
[4.8716933e-01 1.2899101e-05 3.9537350e-04 3.2363136e-03 9.5566381e-03
2.7824407e-03 3.5642386e-06 9.6817694e-06 4.9683377e-01]
loop took 0.18854308128356934 seconds
[2.11263314e-01 3.62360472e-04 7.79911643e-03 1.40771437e-02
3.16963047e-02 3.56153352e-03 1.10482506e-04 2.64724600e-04
7.30865002e-01]
loop took 0.17852258682250977 seconds
[5.9562045e-01 9.7622215e-06 1.7433101e-04 4.6519623e-03 6.2285825e-03
5.1633441e-03 2.7079823e-06 8.0888603e-06 3.8814080e-01]
loop took 0.1807417869567871 seconds
[6.3613790e-01 1.7045422e-06 5.9293208e-05 1.5150562e-03 3.3687411e-03
2.2167012e-03 4.4992106e-07 1.3759092e-06 3.5669878e-01]
loop took 0.1904902458190918 seconds
[3.0532751e-02 8.4837615e-07 5.2463605e-05 3.2035466e-03 1.5352354e-04
1.7064769e-04 1.9033013e-07 8.1629645e-07 9.6588522e-01]
loop took 0.1850905418395996 seconds
[7.1698409e-01 1.7534451e-06 6.5295833e-05 1.0630204e-03 5.0858464e-03
2.1770666e-03 4.6226273e-07 1.3446873e-06 2.7462104e-01]
loop took 0.18003630638122559 seconds
[8.0704749e-01 9.2542467e-07 6.8584241e-06 3.1905512e-03 9.2617294e-04
1.0071040e-02 2.5829220e-07 9.5806251e-07 1.7875572e-01]
loop took 0.1715245246887207 seconds
[8.1627005e-01 4.4470748e-05 9.4758798e-05 1.2970498e-02 7.0062927e-03
5.0863486e-02 1.6212694e-05 4.4946450e-05 1.1268921e-01]
loop took 0.16356348991394043 seconds
[3.2146253e-02 1.0893563e-03 1.7315780e-01 1.1867531e-04 7.5277317e-01
9.7298711e-05 3.5627728e-04 4.0151938e-04 3.9859682e-02]
loop took 0.19198083877563477 seconds
[1.9546534e-01 4.0988565e-05 2.5790289e-02 2.0453426e-04 2.3989089e-01
8.8207795e-05 1.0038092e-05 1.7834725e-05 5.3849179e-01]
loop took 0.1715404987335205 seconds
[3.5563463e-01 2.1260867e-06 5.7028746e-03 3.9074412e-06 5.4808456e-01
1.4182099e-05 4.6471649e-07 6.5600562e-07 9.0556547e-02]
loop took 0.1795198917388916 seconds
[9.9530822e-01 5.2760685e-09 1.7502217e-08 6.2258420e-07 1.6822194e-03
2.9670873e-03 1.9720718e-09 3.6419889e-09 4.1897580e-05]
loop took 0.18251276016235352 seconds
[9.6283674e-01 2.2122357e-07 1.5072217e-07 3.8464263e-05 1.8021752e-03
3.5076745e-02 9.8603479e-08 2.0506954e-07 2.4525329e-04]
Ln: 5 Col: 0

Figure 13

Tensorboard outputs:

Accuracy of the model with respect to steps:

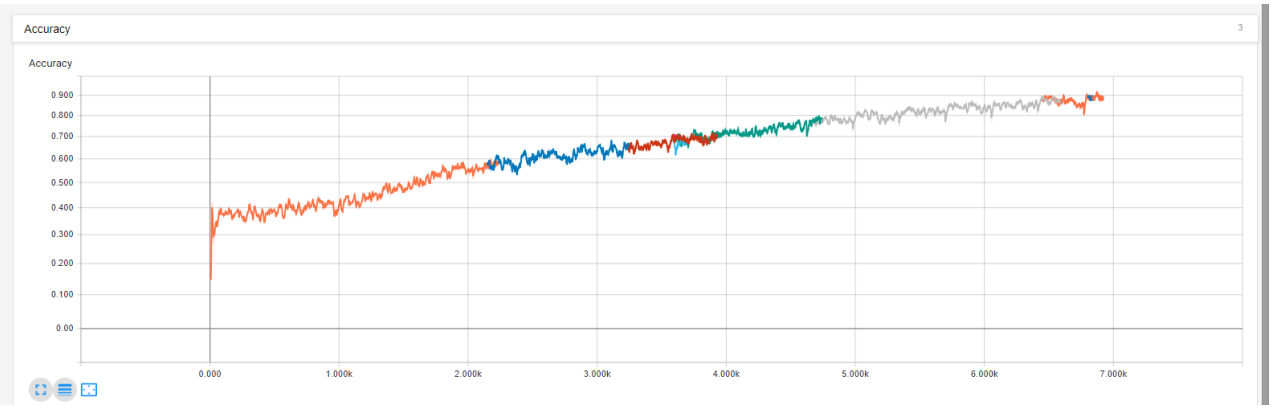


Figure 14

Accuracy/Validation:

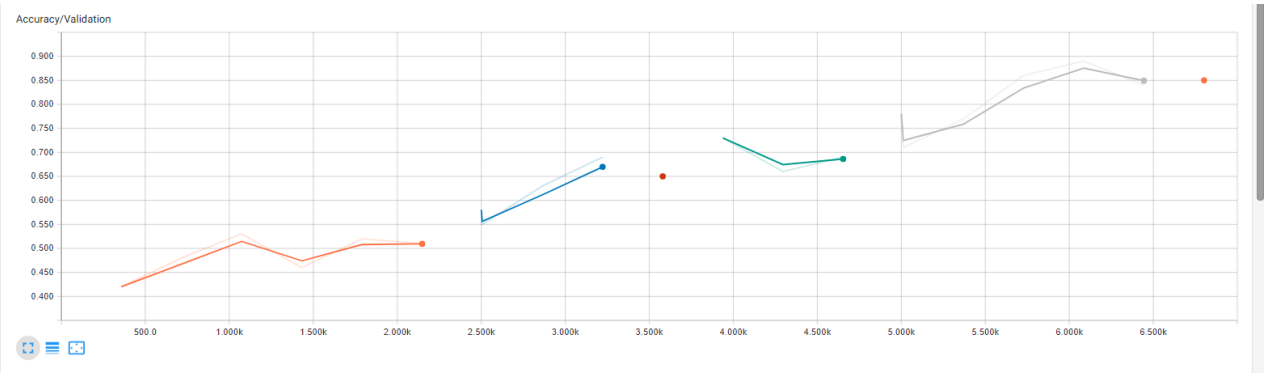


Figure 15

Loss:

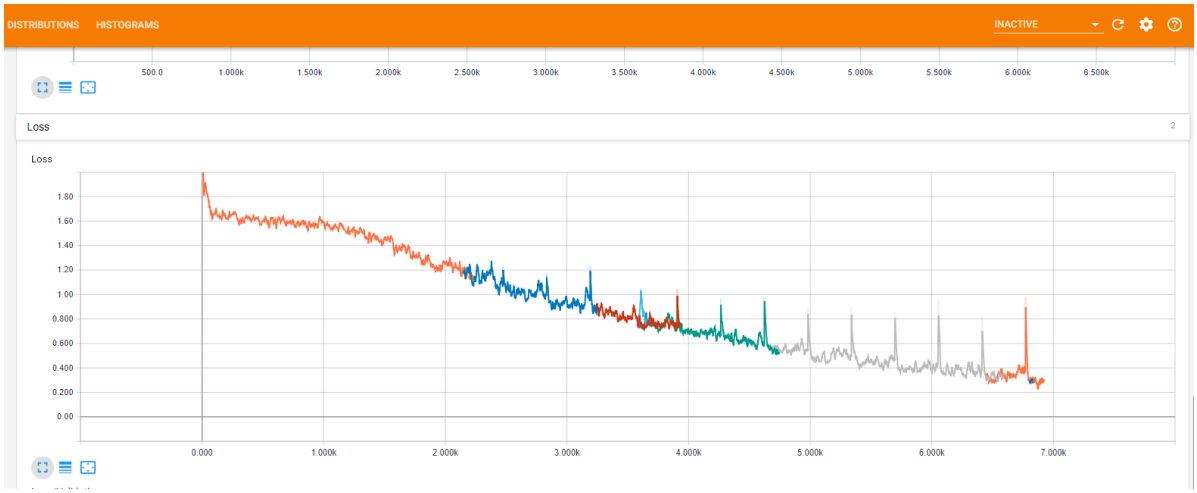


Figure 16

Loss/Validation:

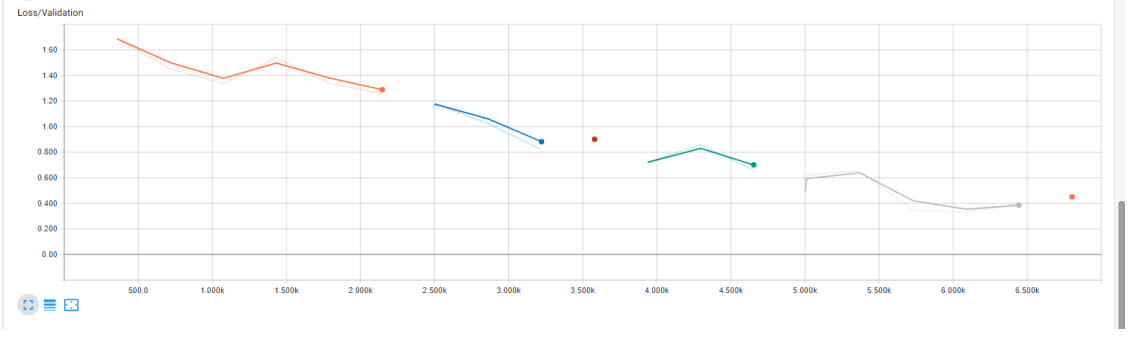


Figure 17

Momentum:

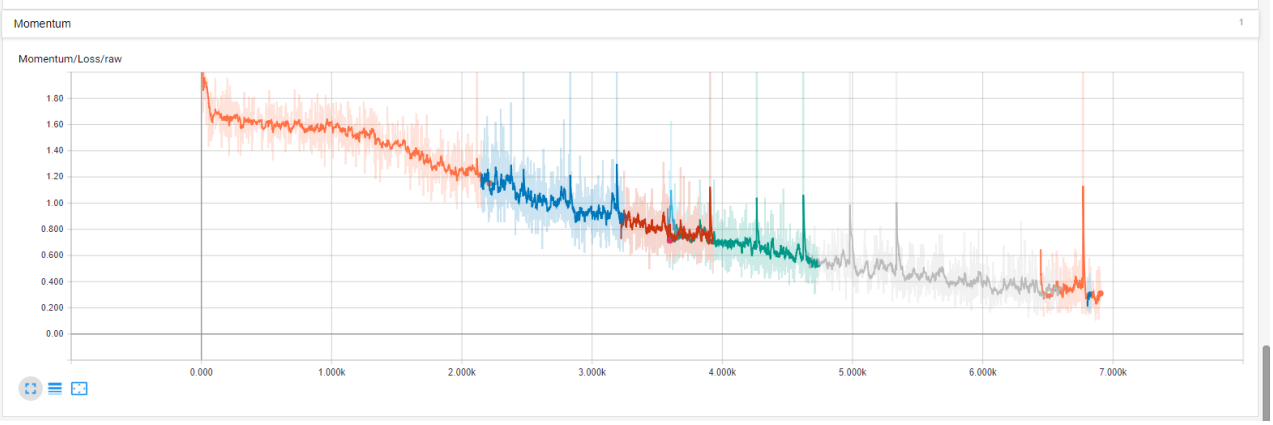


Figure 18

3.CONCLUSION

Currently the deep convolutional neural networks has an accuracy of 89 percent. The model was trained on an i3 7100U CPU for 100 hours on training data which is 518 megabytes long. The accuracy of the prediction can be increased by increasing the amount of data and using high end gpu like RTX 2080 ti, GTX Titan etc,.

The project can be improved further by adding object detection, traffic light detection etc. The interesting thing about the approach used in the project is that this approach can be used for any real world or virtual applications.

APPENDICES

Appendix 1: **Python Tensorflow**

TensorFlow is an open source library for fast numerical computing.It was created and is maintained by Google and released under the Apache 2.0 open source license. The API is nominally for the Python programming language, although there is access to the underlying C++ API.Unlike other numerical libraries intended for use in Deep Learning like Theano, TensorFlow was designed for use both in research and development and in production systems, not least RankBrain in Google search and the fun DeepDream project.It can run on single CPU systems, GPUs as well as mobile devices and large scale distributed systems of hundreds of machines..

Appendix 2: **Keras**

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key

to doing good research.Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU..