Self Driving Simulation using AI

A self-driving car (also known as an autonomous car or a driverless car) is a vehicle that is capable of sensing its environment and navigating without much human input. Such autonomous vehicles detect obstacles that come on its way and analyze the obstacles to make proper decision for its movement. Based upon environmental situation, the autonomous vehicle can move or drive on its own as manually driven by human. We propose to develop a python program that drives a vehicle in a virtual environment for the purpose of creating self driving car. We will simulate the driving in a popular game called Grand Theft Auto (GTA). The main objectives of this project are to detect the objects that come on the path of car, avoid collision and to detect lanes and drive accordingly. We will probably read frames directly by capturing the game's window, rather than working with the game's code itself. It can work with other similar games. However, we chose this game due to it's realistic environment.

To take driving decisions, we will construct a neural network. A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. We will also try to achieve object detection on various objects in the game. The significance of this project is to test the accuracy of driving in a virtual environment which will help us conclude if similar algorithm can be applied in real world.

Submitted by:
Manasi Kattel (18) (9843350494)
Ayush Kumar Shah (44) (9841231569)
CE 4th year

Submitted to:
Santosh Khanal
Assistant Professor

Submission Date:
10th February, 2019

1. Libraries used

1.1. **Keras** : It is a high-level neural networks API, written in Python and capable of running on top of TensorFlow.

1.2. **Numpy** : It is the fundamental package for scientific computing with Python and can be used as an efficient multi-dimensional container of generic data.

1.3. **Pandas** : It takes data (like a CSV or TSV file, or a SQL database) and creates a Python object with rows and columns called data frame that looks very similar to table in a statistical software.

1.4. **Tensorflow-gpu**: It performs highly parallelised computation. It is basically a mini supercomputer running in your PC.

1.5. **opencv-python**
OpenCV (Open source computer vision) is a library of programming functions mainly aimed at real-time computer vision.This library was used to manipulate the frames of the game window.

1.6. **Random** : It generates a random float uniformly in the semi-open range [0.0, 1.0)]

1.7. **Ctypes** : This library was used to take the keys input while generating training data.

2. Methodology

2.1. Dataset:

First, the game screen was obtained using opencv and the frames in the game screen were converted to gray scale. The dataset was generated by driving a bike in third person camera mode for 15 hours in the highway in the game GTA 5 and recording the input keys corresponding to each frame. The possible input keys along with their respective actions are:

1. W- Go forward
2. A- Go left
3. D- Go right

The possible combinations of keys input by the player are:-

A = [1,0,0]
W= [0,1,0]
D = [0,0,1]

Finally, a dataset of 70,000 frames was created, which were stored as numpy arrays of size 160x120 as inputs along with the corresponding keys pressed as output.
The dataset was stored in the format below:

```
array([[array([[ 28,  28,  28, ...,  23,  23,  35],
       [ 28,  28,  28, ...,  23,  23,  30],
       [ 28,  28,  28, ...,  24,  24,  36],
       ...,
       [109, 112, 109, ..., 156, 182,  99],
       [ 19,  40,  51, ...,  97, 101, 111],
       [ 25,  47,  66, ...,  98,  91, 105]], dtype=uint8),
        list([0, 1, 0])],
       [array([[ 28,  28,  27, ...,  23,  23,  35],
       [ 28,  28,  28, ...,  23,  23,  30],
       [ 28,  28,  28, ...,  23,  23,  36],
       ...,
       [112, 108, 108, ..., 148, 201,  99],
       [ 19,  40,  51, ...,  97, 101, 111],
       [ 25,  47,  66, ...,  98,  91, 105]], dtype=uint8),
        list([0, 1, 0])], …..
```

2.2. Preprocessing of datasets

The dataset was unbalanced containing 60770 frames for Go forward, 5080 frames for Go right and 4150 frames for Go left. Since we press W most of the time, there are a large number of frames for Go Forward. If we proceed with this unbalanced data, the neural network will learn to go forward most of the time which is not accurate. So we need to balance the data. We do so, all output classes were reduced to same number of data.

After preprocessing, there were 4117 frames for each of the three classes. So, there were total 12,351 data which was shuffled and divided into training and test dataset in each epoch.

Training set: 1-11,351
Test set: 11,352-12,351

2.3. Building a Convolutional Neural Network:

A simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer.

2.3.1 Convolutional Layer : It is the core building block of a Convolutional Network that does most of the computational heavy lifting. Convolution operations are performed in this layer using a filter.

Convolution operation:

$$(I * h)(x, y) = \int_0^x \int_0^y I(i, j).h(x - i, y - j)didj$$
$$= \int_0^x \int_0^y I(x - i, y - j).h(i, j)didj$$

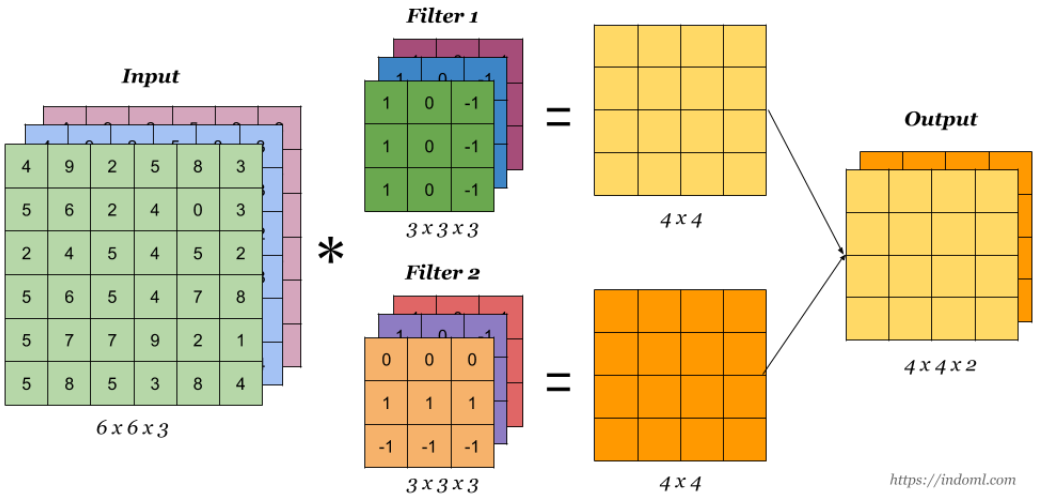

Fig 2.1: Convolution operation

2.3.2 Pooling Layer : Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations.

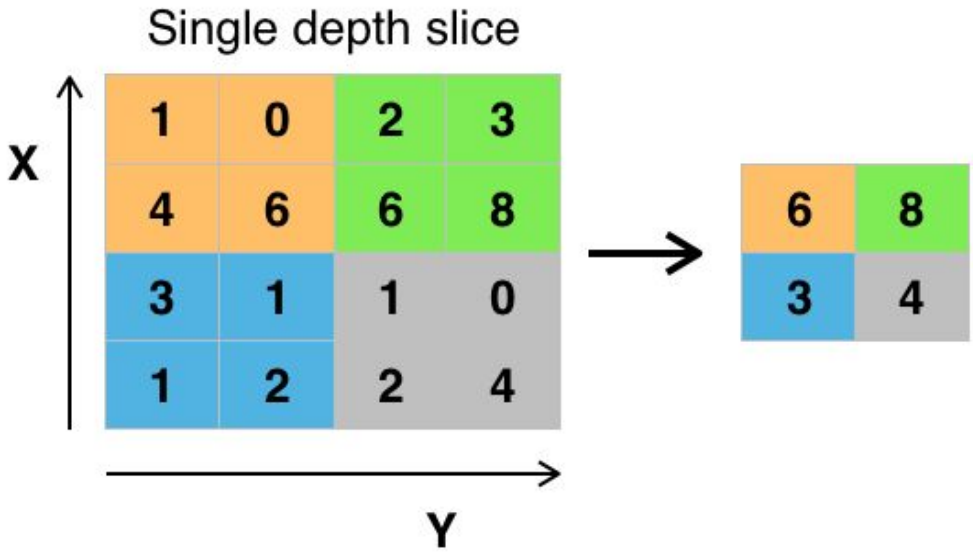

Fig 2.2: Pooling operation

2.3.3 Fully-connected layer : Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

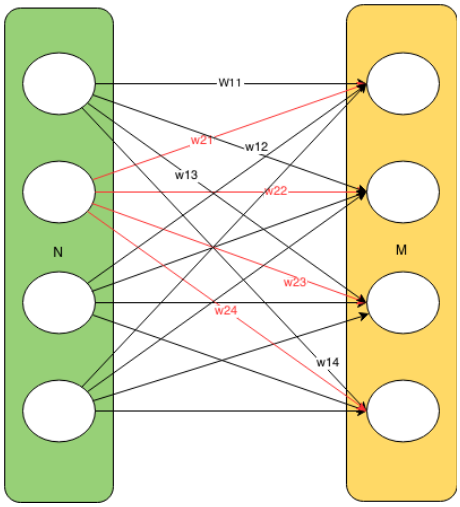

Fig 2.3: Fully connected layer

2.3.4 Batch Normalization

Batch normalization is a method we can use to normalize the inputs of each layer in order to fight the internal covariate shift problem.

During training time, a batch normalization layer does the following:

1. Calculate the mean and variance of the layers input.

   Batch statistics for step 1 is:-

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{Batch mean}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \qquad \text{Batch variance}$$

2. Normalize the layer inputs using the previously calculated batch statistics.

$$\overline{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

3. Scale and shift in order to obtain the output of the layer.

$$y_i = \gamma \overline{x_i} + \beta$$

Where, γ and β are **learned during training** along with the original parameters of the network. So, if each batch had m samples and there where j batches, the inference formulas are:

$$E_x = \frac{1}{m}\sum_{i=1}^{j} \mu_B^{(i)} \qquad \text{Inference mean}$$

$$Var_x = \left(\frac{m}{m-1}\right)\frac{1}{m}\sum_{i=1}^{j} \sigma_B^{2^{(i)}} \qquad \text{Inference variance}$$

$$y = \frac{\gamma}{\sqrt{Var_x + \epsilon}}x + \left(\beta + \frac{\gamma E_x}{\sqrt{Var_x + \epsilon}}\right) \qquad \text{Inference scaling/shifting}$$

2.4. Other Operations performed in our CNN

2.4.1 ReLU Activation function

The Rectified Linear Unit is the most commonly used activation function in deep learning models. The function returns 0 if it receives any negative input, but for any positive value x it returns that value back. So it can be written as f(x) = max(0,x).

Graphically, it looks like this

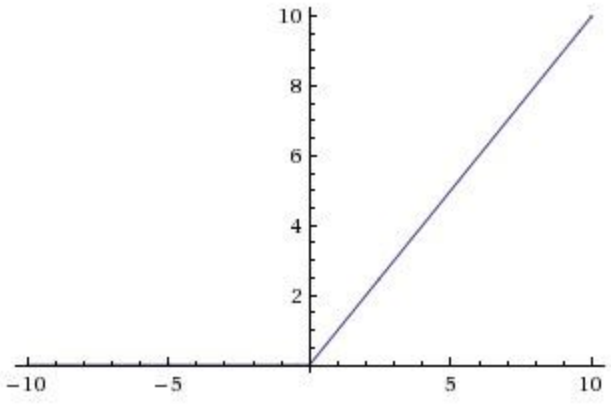

Fig 2.4 ReLU function

It's surprising that such a simple function (and one composed of two linear pieces) can allow your model to account for non-linearities and interactions so well. But the ReLU function works great in most applications, and it is very widely used as a result.

2.4.2 Dropout

Dropout is a technique where randomly selected neurons are ignored during training. They are "dropped-out" randomly. This means that their contribution to the activation of downstream neurons is temporarily removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

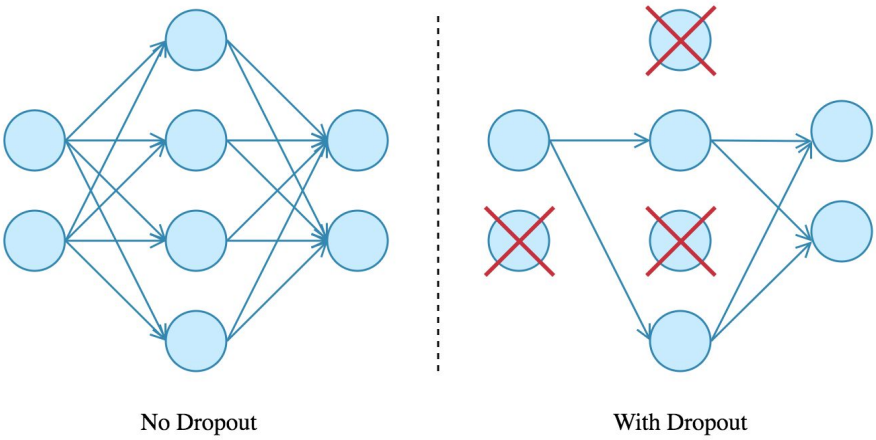

Fig 2.5: Dropout layer

2.4.3 Flattening

Flattening is the process of converting all the resultant 2 dimensional arrays into a single long continuous linear vector. The flattening step is needed so that you can make use of fully connected layers after some convolutional layers.

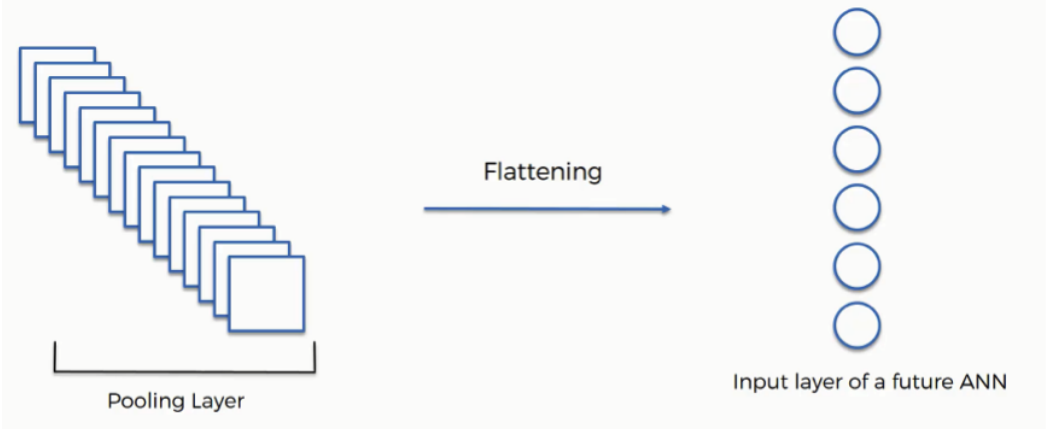

Fig 2.6: Flattening operation

2.4.4 Softmax Activation

Softmax it's a function, not a loss. It squashes a vector in the range (0, 1) and all the resulting elements add up to 1. It is applied to the output scores s. As elements represent a class, they can be interpreted as class probabilities.

The Softmax function cannot be applied independently to each si, since it depends on all elements of s. For a given class si, the Softmax function can be computed as:

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$$

Where sj are the scores inferred by the net for each class in C. Note that the Softmax activations for a class si depends on all the scores in s.

2.5. Neural Network Architecture

The deep convolutional neural network (CNN) architecture proposed in this study is comprised of 3 convolutional layers interleaved with 2 pooling operations, followed by 2 fully connected (dense) layers. The proposed CNN architecture is parameterized as follows:

Layer Type	**Size**	**Number of Kernels**	**Number of Parameters**
Image Input	**120 X 160 X 1**		
Convolution	**11 X 11 X 1**	**96**	11712
ReLU			
Pooling			
Batch Normalization			
Convolution	**5 X 5 X 96**	**256**	614656
ReLU			
Pooling			
Batch Normalization			
Convolution	**3 X 3 X 256**	**384**	885120
ReLU			
Convolution	**3 X 3 X 384**	**384**	1327488
ReLU			
Convolution	**3 X 3 X 384**	**256**	884992
ReLU			
Pooling			
Batch Normalization			
Fully Connected			1052672
tanh			
Dropout			16781312
Fully Connected			
tanh			

Dropout			
Fully Connected			12291
SoftMax			
Classification			

Total params: 21,572,675
Trainable params: 21,571,459
Non-trainable params: 1,216

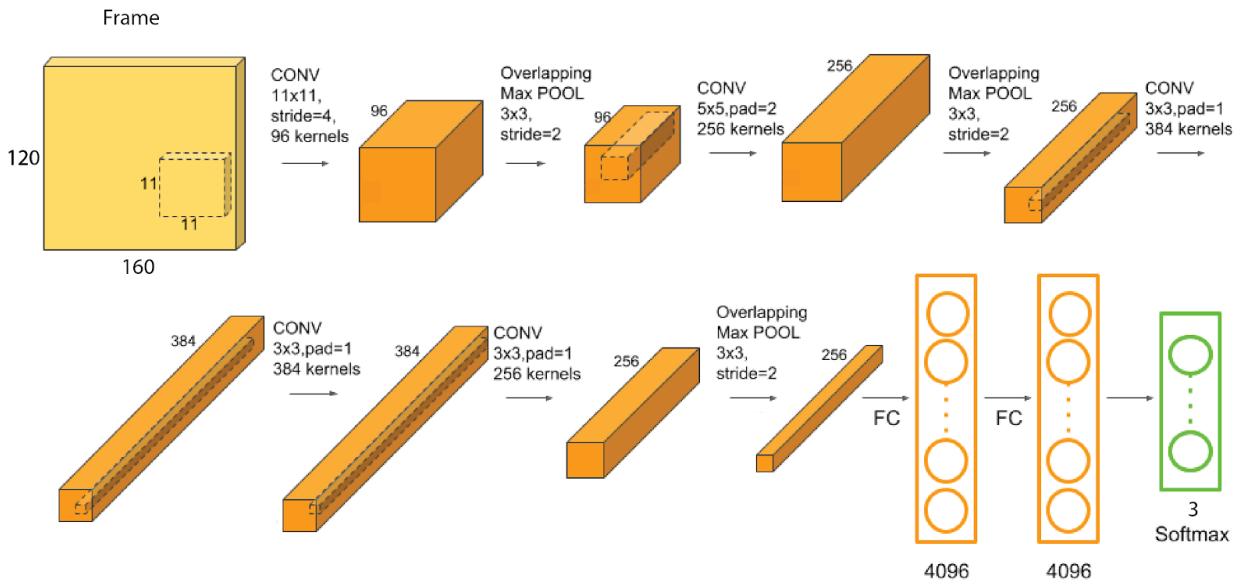

Fig 2.7: CNN Architecture

2.6. Training Process

2.6.1 Learning rate:

Learning rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect the loss gradient. It determines the rate at which the machine learns the features in the CNN. For our CNN, we fixed
Learning rate= 0.001

2.6.2 Categorical Cross-Entropy loss

Also called **Softmax Loss**. It is a **Softmax activation** plus a **Cross-Entropy loss**. If we use this loss, we will train a CNN to output a probability over the CC classes for each image. It is used for multi-class classification.

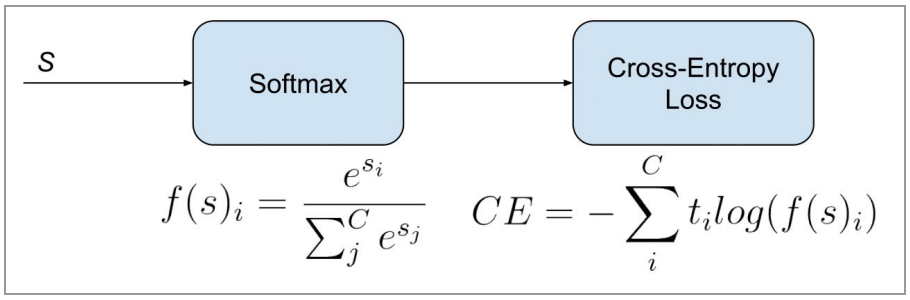

Fig:2.8 Softmax Loss formula

In the specific (and usual) case of Multi-Class classification the labels are one-hot, so only the positive class Cp keeps its term in the loss. There is only one element of the Target vector tt which is not zero ti=tpti=tp. So discarding the elements of the summation which are zero due to target labels, we can write:

$$CE = -log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right)$$

Where **Sp** is the CNN score for the positive class.

2.6.3 Epochs

One Epoch is when an entire dataset is passed forward and backward through the neural network only once. Since, we are using a limited dataset and to optimise the learning and the graph we are using Gradient Descent which is an iterative process. So, updating the weights with single pass or one epoch is not enough. As the number of epochs increases, more number of times the weight are changed in the neural network and the curve goes from underfitting to optimal to overfitting curve.

We trained our datasets for 20 epochs.

2.6.4 Batch size

Since, one epoch is too big to feed to the computer at once we divide it in several smaller batches. Batch Size is the total number of training examples present in a single batch.

For training our dataset, we divided 12351 training examples into batches of 64 (i.e. batch_size=64) and epoch of 20 times.

3. Results and Evaluation

3.1 Performance Metrics used

3.1.1 Confusion Matrix

For a training set of 11,315 data and validation set of 1000 data, the following confusion matrix was obtained on the validation set.

		Predicted Class		
		'A'	**'W'**	**'D'**
Actual Class	**'A'**	**319**	**1**	**0**
	'W'	**0**	**346**	**2**
	'D'	**0**	**3**	**329**

3.1.2 Classification Accuracy

Classification Accuracy is what we usually mean, when we use the term accuracy. It is the ratio of number of correct predictions to the total number of input samples.
Validation accuracy refers to accuracy on validation (test) dataset.

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

After 20 epochs, the following accuracy were obtained on 11,315 training dataset and 1000 validation set:

Training accuracy: 0.988415599
Validation accuracy: 0.969999969

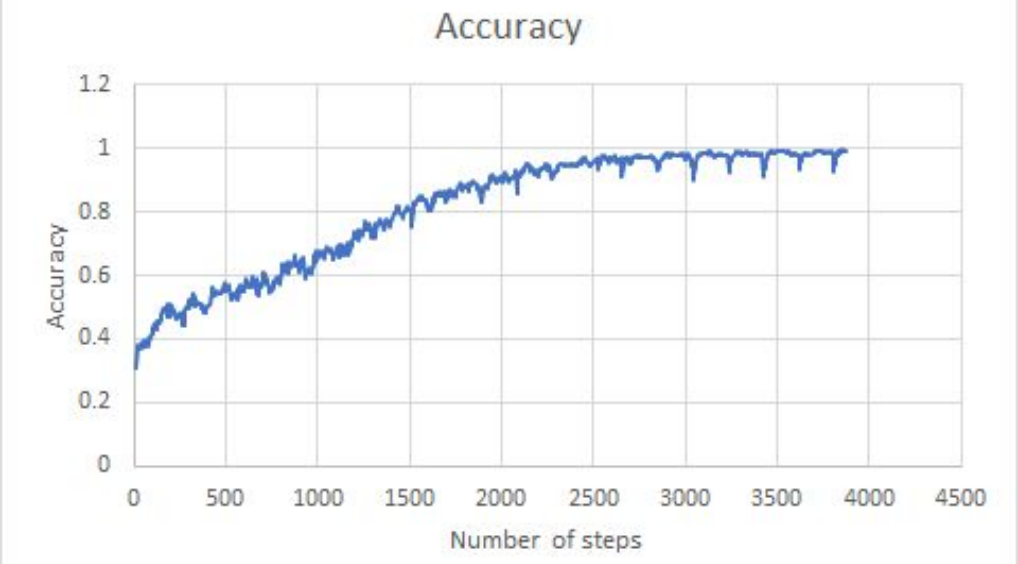

Fig 3.1: Graph of training accuracy increasing with number of steps

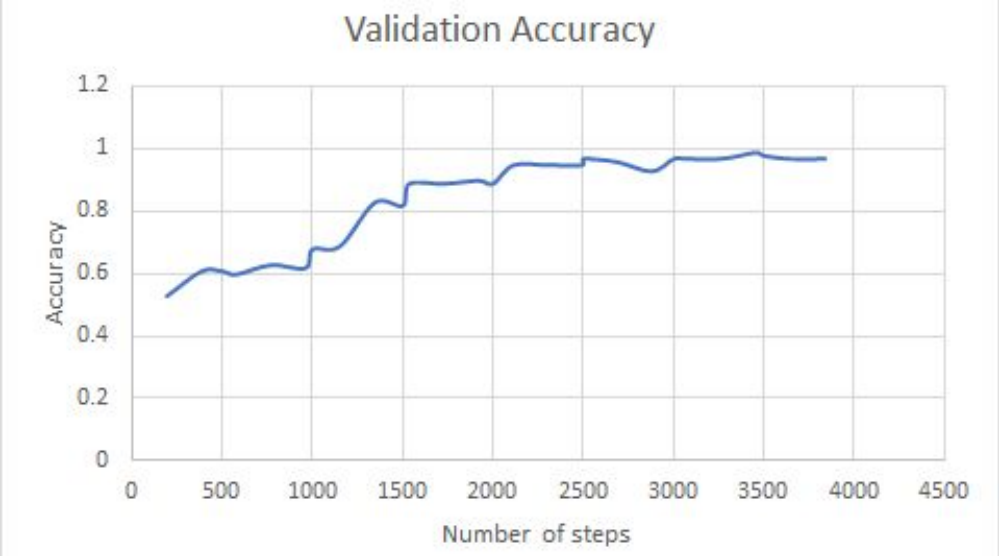

Fig 3.2: Graph of validation accuracy increasing with number of steps

3.1.3 Loss:

A scalar value that we attempt to minimize during our training of the model. The lower the loss, the closer our predictions are to the true labels. We used categorical crossentropy loss function during the training.

After 20 epochs, the following loss values were obtained on 11,315 training dataset and 1000 validation set:

Training loss: 0.029005796
Validation loss:0.163775355

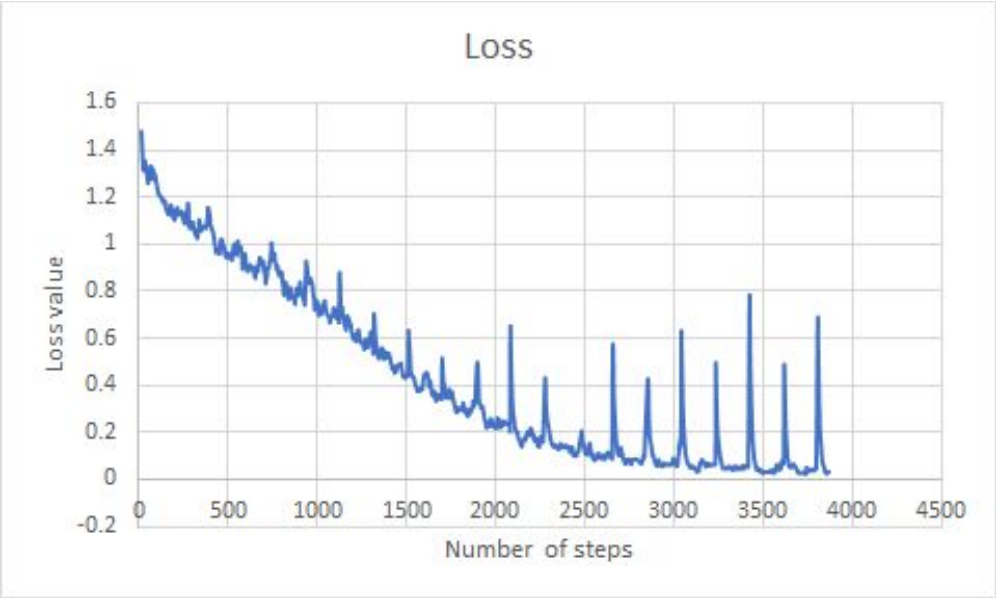

Fig 3.3: Graph of loss value decreasing with number of steps

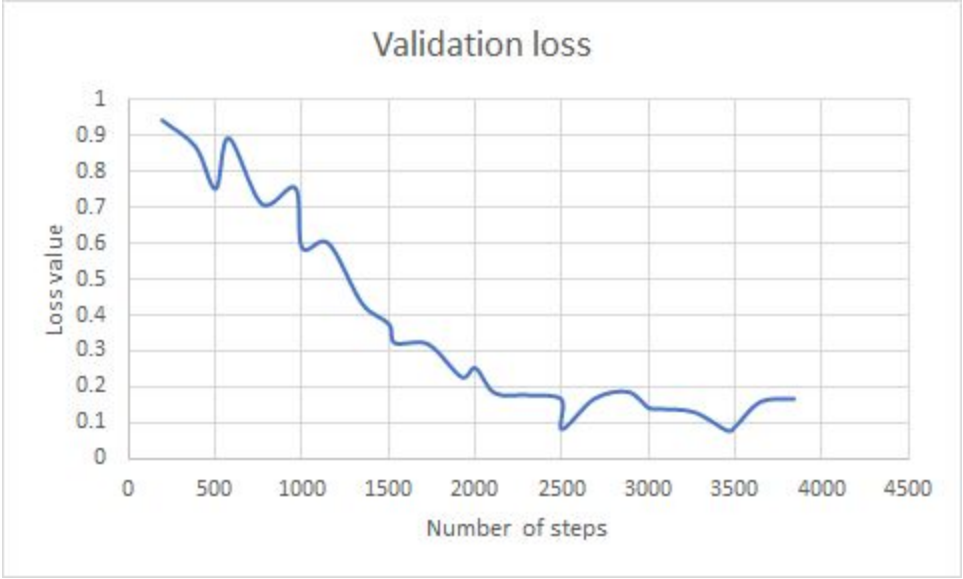

Fig 3.4: Graph of validation loss value decreasing with number of steps

Screenshots:

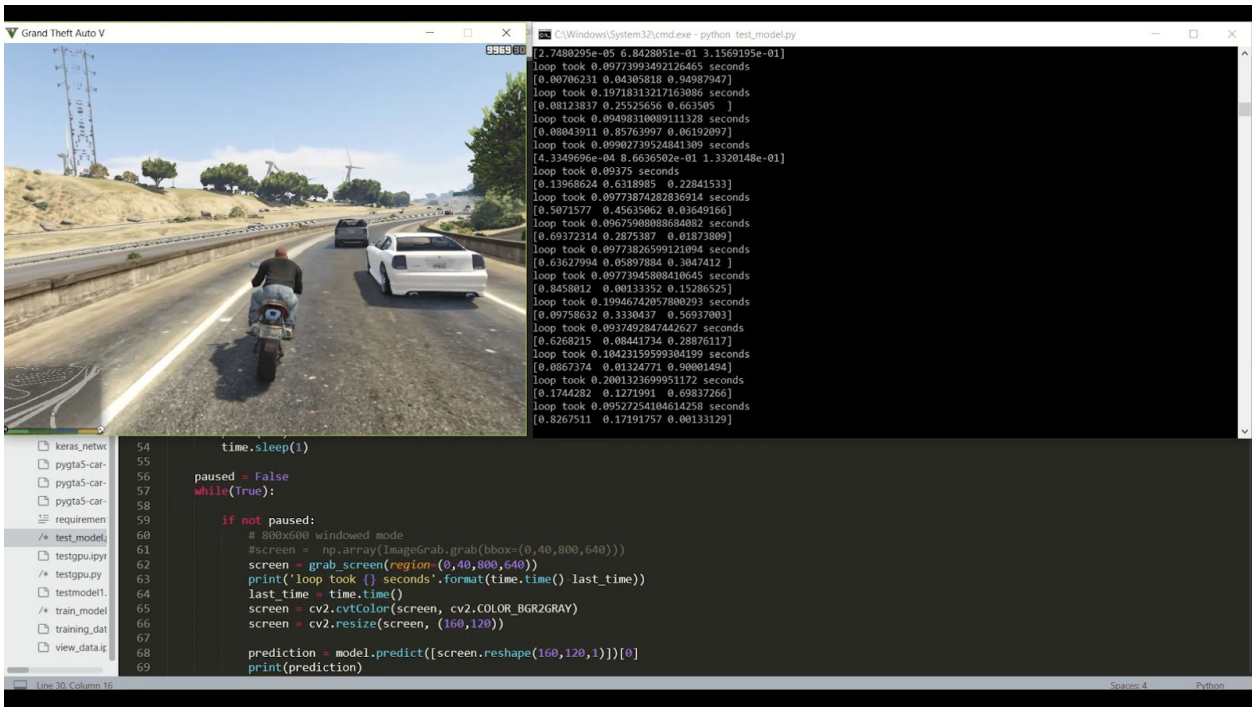

Fig 3.5: Screenshot of AI agent riding a bike in GTA 5

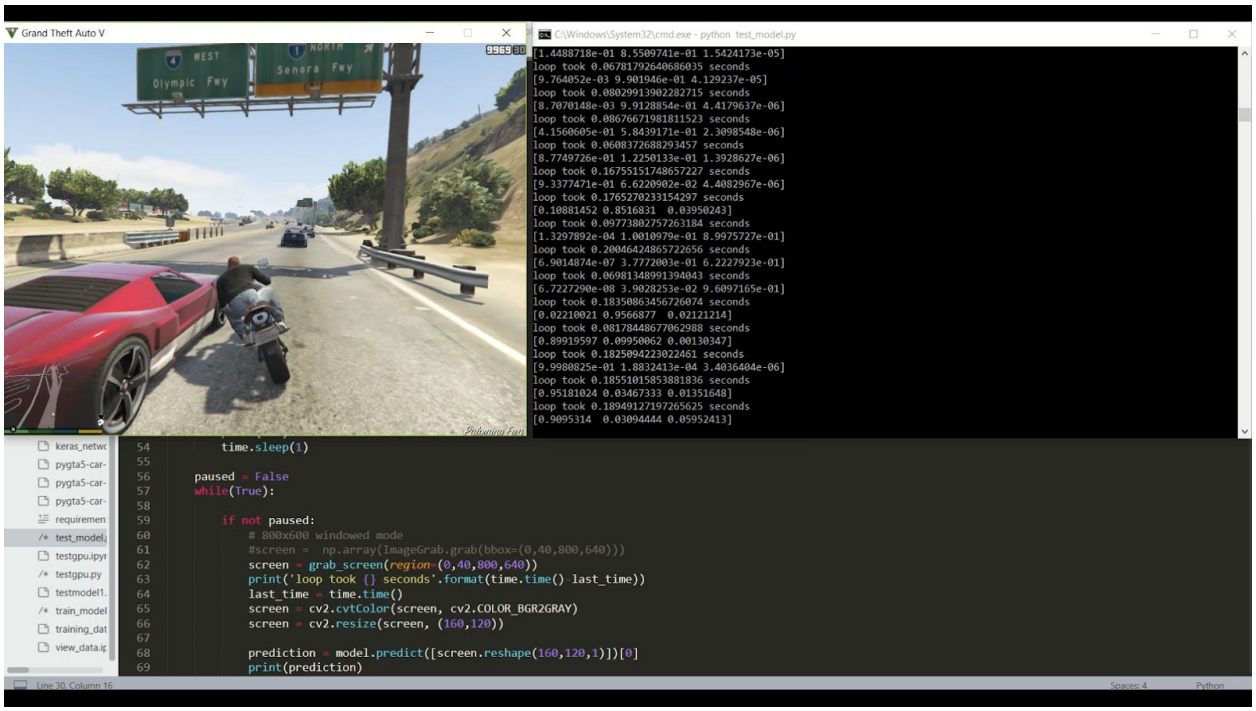

Fig 3.6: Screenshot of AI agent overtaking a car in GTA 5

4. Limitations

- The agent still crashes with other vehicles when driving at high speed.
- The agent sometimes goes off road.
- The agent is not able to drive during night time.
- The AI agent cannot reverse or slow down.
- The agent can drive only in the highway not in the crowded cities.

5. Conclusion and Future Plans

Hence, in this project, we built a deep convolutional neural network architecture to train self driving vehicle in a gaming environment GTA 5. We obtained satisfiable results as the AI agent was able to learn to drive but not perfectly. To overcome the limitations mentioned above, the training process should be made more effective by including the combination keys such as WA, WD, SA, SD, and reverse (S) as output classes and retraining the CNN again. Likewise, the accuracy can be increased by generating larger training data by playing the game for more hours including the cities and also playing during the night time.

References:

[1]"Python Programing Tutorials", *Pythonprogramming.net*, 2019. [Online]. Available: https://pythonprogramming.net/testing-self-driving-car-neural-network-python-plays-gta-v/?fbclid=IwAR23rw2GKNEC0S2SdujSO_4j9LE3_BhToC9tg0O9KC2Pk804zLfGtGjNEng. [Accessed: 09- Feb- 2019].

[2]S. Nayak, "Understanding AlexNet | Learn OpenCV", *Learnopencv.com*, 2019. [Online]. Available: https://www.learnopencv.com/understanding-alexnet/. [Accessed: 09- Feb- 2019].

[3]"Convolutional Neural Networks - The Math of Intelligence (Week 4)", *YouTube*, 2019. [Online]. Available: https://www.youtube.com/watch?v=FTr3n7uBIuE&t=369s. [Accessed: 09- Feb- 2019].

[4]"Batch normalization: theory and how to use it with Tensorflow", *Towards Data Science*, 2019. [Online]. Available: https://towardsdatascience.com/batch-normalization-theory-and-how-to-use-it-with-tensorflow-1892ca0173ad. [Accessed: 09- Feb- 2019].

[5]"Epoch vs Batch Size vs Iterations – Towards Data Science", *Towards Data Science*, 2019. [Online]. Available: https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9 . [Accessed: 09- Feb- 2019].

[6]"Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names", *Gombru.github.io*, 2019. [Online]. Available: https://gombru.github.io/2018/05/23/cross_entropy_loss . [Accessed: 09- Feb- 2019].

[7]"Simple Image Classification using Convolutional Neural Network — Deep Learning in python.", *Becoming Human: Artificial Intelligence Magazine*, 2019. [Online]. Available: https://becominghuman.ai/building-an-image-classifier-using-deep-learning-in-python-totally-from-a-beginners-perspective-be8dbaf22dd8. [Accessed: 09- Feb- 2019].

[8]J. Brownlee, "A Gentle Introduction to the Rectified Linear Activation Function for Deep Learning Neural Networks", *Machine Learning Mastery*, 2019. [Online]. Available: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks. [Accessed: 09- Feb- 2019].

[9]"CS231n Convolutional Neural Networks for Visual Recognition", *Cs231n.github.io*, 2019. [Online]. Available: http://cs231n.github.io/convolutional-networks/. [Accessed: 09- Feb- 2019].