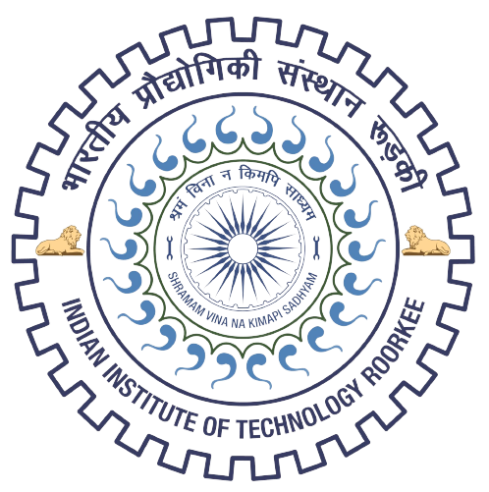
**CSN – 382**

*Machine Learning*

*Mini-Project-Report*



Submitted by:

|  |  |
| --- | --- |
| **Name:** | **Enrollment:** |
| Anshul Alok Gharde | 20114013 |
| Sagiraju Ram Prabhas Varma | 20114082 |
| Tharuniswer R | 20114100 |

Contents

[1. Problem Statement: 2](#_Toc132790773)

[2. Implementation: 3](#_Toc132790774)

[2.1. Early ideas (lane detection) 3](#_Toc132790775)

[2.2. Generating training data 5](#_Toc132790776)

[2.3. Balancing training data/ data augmentation 6](#_Toc132790777)

[3. Models explored: 6](#_Toc132790778)

[3.1. GoogLeNet 6](#_Toc132790779)

[3.2. Alexnet 7](#_Toc132790780)

[4. Training on modified Alexnet (Alexnet2) 10](#_Toc132790781)

[4.1. Params: 10](#_Toc132790782)

[4.2. Training set 10](#_Toc132790783)

[4.3. Evaluation: 10](#_Toc132790784)

[4.4. Testing on the simulation 11](#_Toc132790785)

[5. Some limitations: 11](#_Toc132790786)

[5.1. Trained on traffic-less road 11](#_Toc132790787)

[5.2. Low vehicle speed 11](#_Toc132790788)

[5.3. Better on bike 12](#_Toc132790789)

[5.4. Different vehicles, different parameters!!! 12](#_Toc132790790)

[5.5. Performance on white roads 12](#_Toc132790791)

[6. Future implementations on scaling model 12](#_Toc132790792)

[7. Conclusion 13](#_Toc132790793)

# Problem Statement:

To make an Autonomous Driving System in python which drives on the roads of GTA-V, an interactive computer game.

# Implementation:

## Early ideas (lane detection)

Our early idea before making the model was to first test if the program can run on the roads of GTA-V by detecting Lanes. No machine learning algorithm was used.

Thus, in each iteration of a loop, we tested if the lanes can be generated. For that, in each iteration we took a screenshot of the game’s screen and pre-processed it using OpenCV to convert it to grayscale and focusing on just the region of interest. The program was generating 10+ frames per second, which was a good result to work on.

A person riding a motorcycle down a road

Description automatically generated with medium confidence 

This gray scaled image is then blurred using Gaussian Blur such that the dominant line will get highlighted.

The region of interest is just the part below the horizon and in the polygon-area shown below.

A picture containing text, dark, night sky

Description automatically generated A picture containing text, dark

Description automatically generated

After that, we used line function to draw those lines. The results are imposed on the original images as given below.

A person riding a motorcycle on a road

Description automatically generated with low confidence 

After this, the program was working a little bit moderately, but it showed us that if a program can run by using lane detection, a good CNN model can also do the same when given the screenshot of game’s screen as input because both work on the principle of extracting the data.

## Generating training data

The dataset for the model is generated by playing the game for hours and simultaneously taking screenshot, not manually along with the corresponding hot-vector of the choice, i.e., LEFT = [ 1, 0, 0], STRAIGHT = [ 0, 1, 0], RIGHT = [ 0, 0, 1]. The Original images (size 800x600) are converted to gray-scale, and resized (to size 160x120). The images and their corresponding choice are stored in a .npy file.

We generated a total of 22 .npy file with each file. Looking at one of the files, we got to know that there are 958 images for LEFT, 10488 images for STRAIGHT and 1025 images for RIGHT. This unevenness of the data is due to the route we used to train the data as shown in picture:



Figure 1: The purple road is the training route

## Balancing training data/ data augmentation

Since most of our data is labelled straight, the dataset is biased. This could lead to bad prediction if we train the model directly on this dataset. So first we made the number of images same for each choice.

To increase the amount of data for turns, we inverted the images labeled LEFT and then added them to RIGHT, and then inverted all the images labeled RIGHT and saved it as LEFT. Thus, we have more data corresponding to both LEFT and RIGHT,

A person riding a skateboard down a road

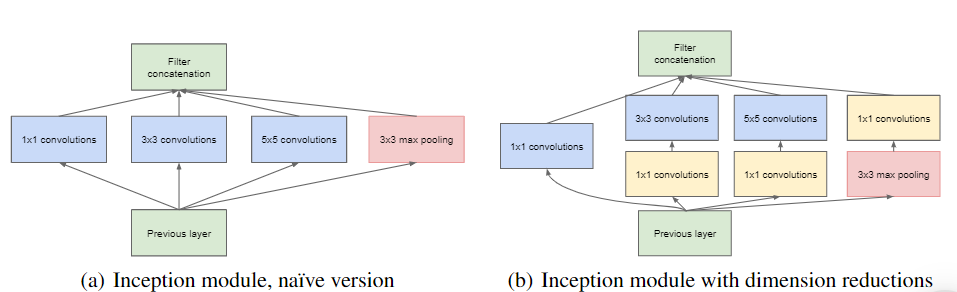
Description automatically generated with medium confidence 

Then we randomly choose len(RIGHT(or LEFT)) number of images from STRAIGHT and saved it. Thus, our data has the same number of images for each category. Doing this for all the 22 .npy files, we have total of 3,05,805 images. (101935 images of STRAIGHT, LEFT and RIGHT each)

# Models explored:

## GoogLeNet

### Architecture of GoogleNet

The overall architecture is 22 layers deep. GoogLeNet is a type of convolutional neural network based on the Inception architecture. "inception modules" are small sub-networks that perform parallel convolutions with different filter sizes. It utilises Inception modules, which allow the network to choose between multiple convolutional filter sizes in each block. It uses many kinds of methods such as 1×1 convolution, global average pooling and not using fully connected layers to decrease the number of parameters (weights and biases) of the architecture. By reducing the parameters, we also increase the depth of the architecture. 

### Advantages

1. It can reduce the number of parameters by 1×1 convolution, global average pooling etc. which reduces the computational resources required.
2. Usage of inception modules allows the network to capture features at multiple scales and resolutions.

### Disadvantages

1. The architecture of GoogleNet is more complex than alexnet which makes it harder to train and requires more computational resources than alexnet.
2. The use of multiple parallel convolutions in the inception modules can lead to increased memory usage and computational overhead.

### GoogleNet’s complexity:

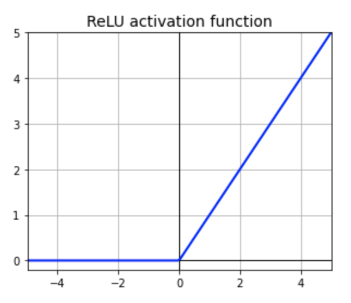
With our limited training data on hand, GoogleNet appeared complex with its large number of layers. We tried our testdata on GoogleNet initially but it took a whole lot of time on our systems. So we decided to look for a simpler model like AlexNet shown below.

## Alexnet

Alexnet is a convolutional neural network (CNN) developed by Alex Krizhevsky, Geoffrey Hinton, and Ilya Sutskever in 2012. It is the first large-scale convolutional neural network to be used in the field of computer vision. Alexnet is a deep learning model that is used for image classification and object recognition. It is a deep learning model that is composed of several layers of convolutional and pooling layers.

Alexnet is a powerful model that has been used in many applications such as image classification, object detection, and image segmentation. Alexnet has been used in many applications such as facial recognition, autonomous driving, and medical imaging.

A Convolutional Neural Network is a type of artificial neural network that is primarily used for image recognition applications. It consists of multiple layers called perceptron’s for learning features present in images with utmost detail.

 Chart, line chart

Description automatically generated

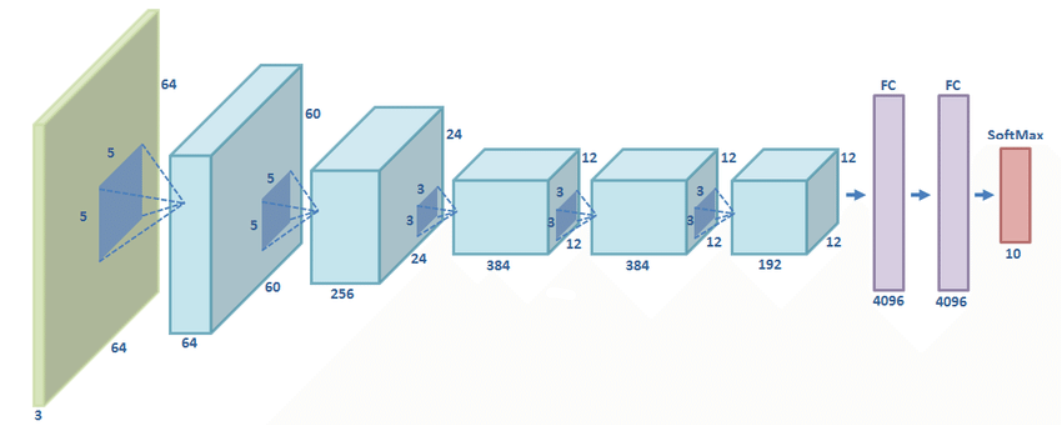
AlexNet consisted of 8 layers and used the ReLu activation function which was a major discovery in deep learning. It got rid of the vanishing gradient problem since now the gradient values were not limited to a certain range. It was the first GPU based CNN model.

### Advantages

* 1. AlexNet was the first major CNN model that used GPUs for training. This leads to faster training of models.
  2. The architecture of AlexNet is relatively simple compared to more recent deep learning models, which makes it easier to train on hardware with limited computational resources while also giving good enough results.

### Disadvantages

1. The architecture of AlexNet is relatively shallow compared to more recent deep learning models, which means that it may not perform as well on more complex tasks.
2. The use of max-pooling layers in AlexNet can lead to loss of information, which may limit its ability to capture fine-grained details in images.



### Architecture of Alexnet

Alexnet is composed of five convolutional layers and 2 fully connected layers. The convolutional layers are used to extract features from the input image. The fully connected layers are used to classify the extracted features. The convolutional layers are composed of convolutional filters, pooling layers, and non-linear activation functions. The pooling layers are used to reduce the size of the feature maps. The non-linear activation functions are used to introduce non-linearity into the model. It also worked well for the time with color images.

The ReLu activation function used in this network has 2 advantages. It does not limit the output unlike other activation functions. This means there isn’t too much loss of features.

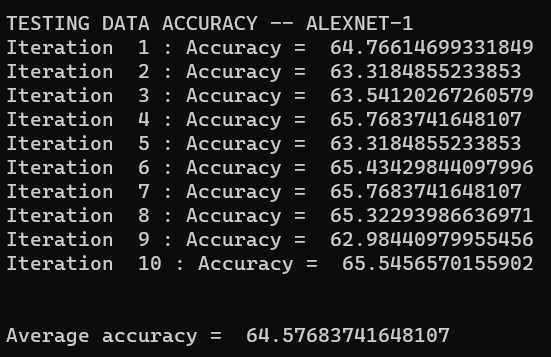
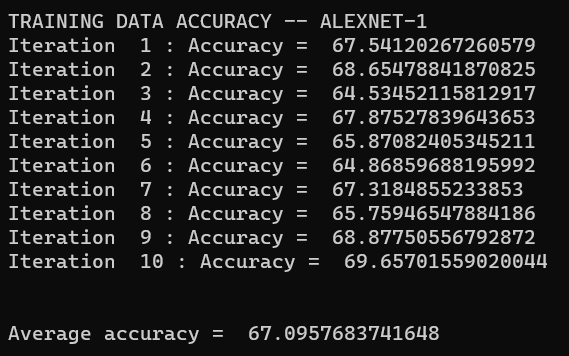
It negates the negative output of summation of gradients and not the dataset itself. This means that it will further improve model training speed since not all perceptrons are active.

### Training of Alexnet

Alexnet is trained using the stochastic gradient descent algorithm. The training process involves adjusting the weights of the model to minimize the loss function. The loss function is used to measure the difference between the predicted output and the actual output. The weights of the model are adjusted using backpropagation. Backpropagation is an algorithm that is used to calculate the gradients of the loss function with respect to the weights of the model.

### Performance of Alexnet in our case

Alexnet has achieved impressive results in image classification and object recognition tasks. It has been used in many applications such as facial recognition, autonomous driving, and medical imaging. In our case when we trained our model on Alexnet the accuracy turned out to be below 70% as seen in the images. This is a clear case of underfitting. We had 3L datapoints, so this underfitting suggests that the model does not have enough layers to train. So we shifted to modifying AlexNet a bit for our purpose.



# Training on modified Alexnet (Alexnet2)

## Params:

Our modified version of Alexnet uses 14 layers out of which there are 1 input layer, 9 convolution layers, 4 fully connected layers and 1 output layer.



The 9 convolution layers are using ‘ReLU’ as their activation function, 4 fully connected layers is using ‘tanh’ and the output layer uses ‘softmax’ function.

The loss function is set to cross-entropy since there is prediction involved in our model that uses probability. We are using a momentum optimizer in our code. It is just an upgrade to stochastic gradient descent in order to smoothen data.

## Training set

The model is trained on a dataset of 3L images, for 18 epochs which took around 6 hours. No further increase in accuracy was seen on the subsequent epochs of the model.

## Evaluation:

For evaluation purposes, first we are checking the accuracy of the model on the random images present in the training dataset, and we got an average of 89.81% accurate model as a result.

Text

Description automatically generated

When we tested the model on new data captured with corresponding label, we got an average of 84.81% accurate model.

## Testing on the simulation

After training the model, to see how it works, we need to run the model on real time data. Refer to the video attached to see the performance.

**Note:** The model has this accuracy of 84-90% but, how good it looks on the simulation depends on the extra parameters which we must set for different vehicles. Some parameters include turn time i.e., time for which each label is applicable which corresponds to time of application of each keyboard key, stop time i.e., time after which brakes are applied to slow down the vehicle etc.

# Some limitations:

## Trained on traffic-less road

The dataset of the model is trained on a NO-TRAFFIC road, such that it can get familiar with the roads, and the structure of the road to make the decision. Thus, we are not able to drive the vehicle on a traffic road as it does not have a stopping criterion.

To tackle this problem, we can add an object detection algorithm to detect vehicles on side and front, and act accordingly.

## Low vehicle speed

The speed of the vehicle is low in order to let the model work good as frame capture rate is constant.

We have kept the speed of the vehicle low, as we need the AI to make a good turn as turning is a function of speed i.e., for a fast-moving vehicle making sharp turns is a necessity, but for slow vehicles, smaller turns will give a good result. Thus, for better result, slow speed is modulated using program.

This can be also solved by adding a mod which can have a track of speed and using that, it can set the turning parameter.

## Better on bike

After doing some testing on different types of vehicles, we found that the model works better on two-wheeler such as bikes, than a four-wheeler, which we think is due to the fact that first, while training we used bike only, thus it’s not performing that well on vehicle other than 2-wheeler. The second reason is the game mechanic, as both cars and bike have different mass, different momentum, and thus car is much harder to control by the model which is trained on the dataset of bike.

To tackle this, what we can do is collect more data using car as a vehicle and train the model to get better result.

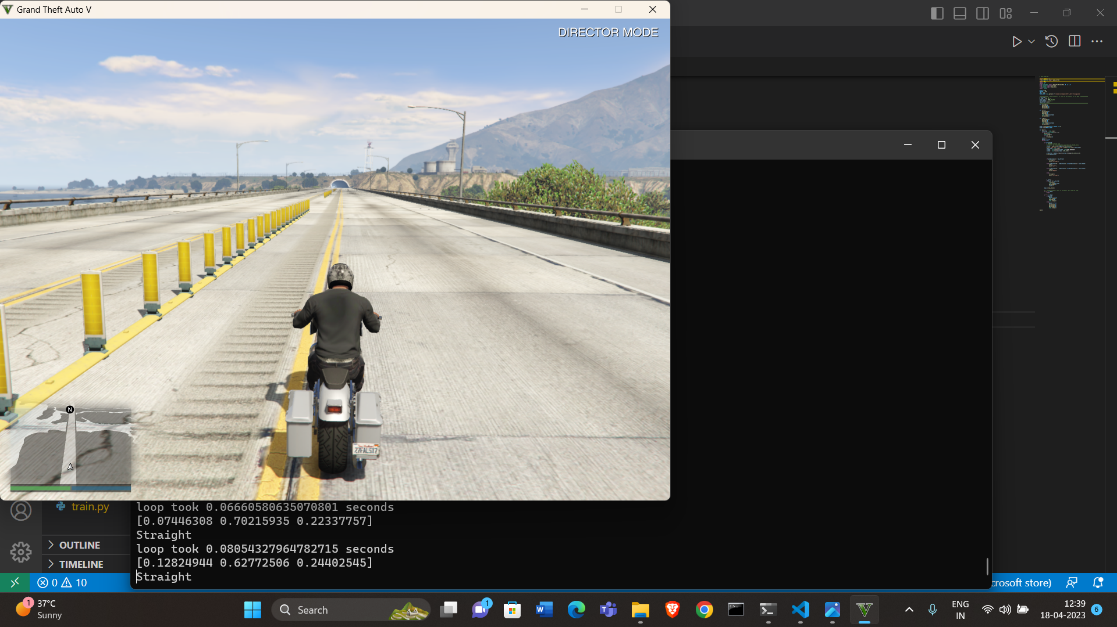
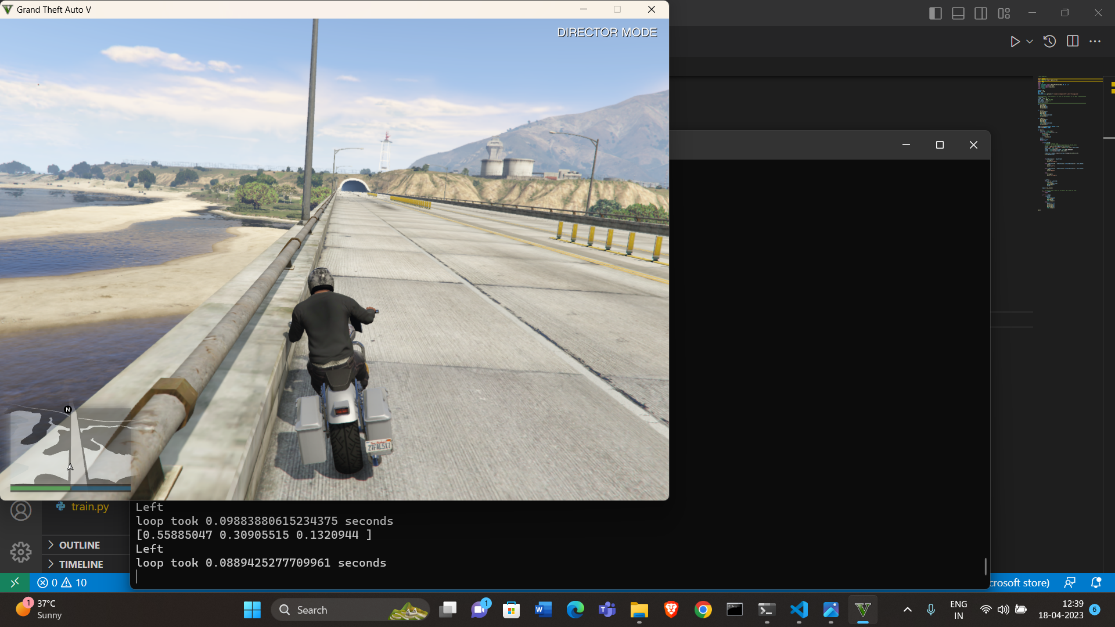
## Different vehicles, different parameters!!!

All the vehicles have different acceleration and handling (turn control), thus to make the run successful, we need to adjust some parameter such that the speed of the vehicle is constant and turns better. Thus, every time we change our vehicle, we also must change these parameters int the control code.

This can be also solved by having a speed tracking mod to control the speed of vehicle and have vehicle details which can be added as parameter to the turn function.

## Performance on white roads

The dataset is obtained by playing the games for hours and capturing screen and the corresponding output. But most of our route has road of black color (coal tar) and there are white-cement roads for bridges. Thus, the bridges in the dataset acts as outliers and due to that, the model sometimes gets confused on bridges and makes abrupt turns and tries to crash on the side.

We can solve this problem as well by obtaining more data on the cemented road.

Other idea is to use object detection so that when the vehicle approaches a wall on the side, it can move away from it.

# Future implementations on scaling model

* To run on a better ML algorithm model. Running on a higher layer network like GoogLeNet requires a lot of training data which can be generated given enough time.
* Using object detection algorithms can be used so that it works on traffic roads too.
* To find a sweet spot between our split in training data. More precisely we have made the number of labelled data that has straight to be equal to the number of left equal to the number of rights. This can lead to a uniformity between left, right and straight which is not desirable as our model mostly is tasked to run on straight roads so a bias towards straight is required but not as much as the ratio present in our training set.
* Some mods can be applied to control speed, limit speed and slow down while turning.
* To train the pretrained model by using reinforcement learning, as now the agent doesn’t have to start from zero, and can stay in lanes, by creating some point system.
* To achieve more accuracy, we can get more training data.

# Conclusion

The above report explains our model in brief. The self-driving model has a decent accuracy for a real time application with the limited training data, suggesting that alexnet is a model that can be a starting point for building complex self-driving models by adding additional layers.