International Journal of Interdisciplinary Innovative Research & Development (IJIIRD)

ISSN: 2456-236X

Vol. 02 Special Issue 05 | 2018

Autonomous Driving Car Using Convolutional Neural Networks

[1]Raj Chaudhari, [2]Shivani Dubey, [3]Jayesh Kathale

ABSTRACT

We propose to work on building a neural network and training it to drive a car autonomously on roads without any human intervention. We make use of Grand Theft Auto 5, an open world game developed by Rock star Games to extract data as frames and simulate final output. The frames are then processed and are fed to the network for training. The network we have trained is an Inception V3 model which uses convolutional neural network. The data we have collected from the game is of exactly 100 Gigabytes of various random road conditions where each frame shows throttle, brake, steering, driving mode & mini map showing the exact location and the path to be followed. The network we have trained can predict steering at an accuracy of 35% despite being trained at only 100 GB of data and a period of only 25 hours with the limited processing power. Although with this accuracy, our trained model runs exceptionally well avoiding most collisions.

Keywords: ***Processing, Neural Networks, Inception, Convolution, Training, Steering.***

I. INTRODUCTION

In this rapidly moving world, cars and other modes of On-road vehicles play a very important role. As we know that, there are billions of on-road vehicles in the world right now. Such huge traffic is also responsible for a humongous number of deaths and injuries because of accidents. A large proportion of reason for these accidents is driver negligence, ignorance of traffic rules and other driver-related mistakes. An Autonomous Driving Car is a possible viable solution to this which help in bringing down the fatalities to a minimal extent. We are using Grand Theft Auto V, a simulation game with incredibly realistic environment as a source of visual data that will be used to train the Car. The main purpose of using GTA V is that it contains 250+ different types of vehicles, thousands of random pedestrians and animals, a lot of different weather conditions, life like roads and bridges, Traffic signals, tunnels. Such large variety of testing conditions will be difficult to find practically. Hence, GTA V is the ideal fictional world for self-driving lessons. All the coding done is in Python language with appropriate libraries as listed later. Artificial Neural Network which is an adaptive technology as it modifies itself from initial training and subsequent runs to provide more information about the respective surrounding environment is been used. Likewise, we train an Inception V3 module from scratch building our own weights and parameters. Inception V3 incorporates Convolutional neural networks for image classification which reduces the requirement for pre-processing as compared to other image classification networks and thus providing with better results & less error rates.

II. NEURAL NETWORKS USED

Convolutional Neural Networks (ConvNets) have been a rage in the recent times amongst modern day research and development community to try and ease various applications right from image processing to pattern recognition and classification. ConvNets are basically a modified version of multi-layer perceptron designed such that they require very less processing as compared various other networks. Being a part of Deep Learning, its architecture contains a humongous number of hidden layers between the input and the output one to provide ultimate optimization. Over the years, Inception architecture has become a huge success for Computer Vision. Being introduced in the year 2014 through the paper, 'Going Deeper with Convolutions',

which was GoogLeNet submission for widely-acknowledged ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [1]. This module proved to better from its two years back ILSVRC winner by using 12 times fewer parameters and still proving to be more accurate. The Inception V3 architecture was later published by Szegedy et al., 'Rethinking the Inception Architecture for Computer Vision' in the year (2015) which is primarily an update to their previous work to further improve the accuracy. This Inception architecture of GoogLeNet was also designed to perform well even under strict constraints on memory and computational budget. Therefore, Inception can be a better alternative as compared to other fellow networks depending on our requirements as we already have limited processing capability and storage constraints [2]. Gradient descent is a Neural Network performance optimization algorithm. Stochastic gradient descent (SGD) performs a parameter update for each input-output training pair. SGD performs an individual parameter update at a time [3] which scales down the redundancy. Thus, it becomes possible to learn SGD online and make it even swifter. Momentum, a method is used to accelerate SGD in the appropriate direction and reduce oscillations. Nesterov momentum allows us to efficaciously look forward by calculating the gradient not respective to our current parameters but with respect to the proximate future position of our parameter.

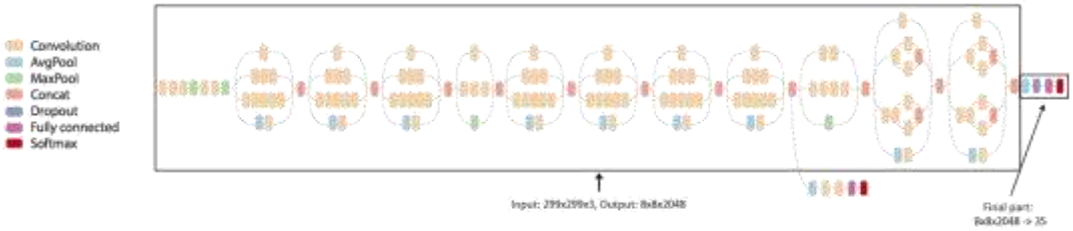

Fig (1): Inception architecture with modified input output layers.

III. RESOURCES USED

A. Hardware

Looking at the requirements to train an Inception model from start, we needed a device which would possess great processing power. So, we tried to get our hands on the best machine we can within our limits. The device which we are using is "MSI GE62VR Apache Pro Notebook" which is powered by a 2.8 GHz Intel Core i7-7700HQ Quad-Core processor and has a DDR4 RAM of 16 GB, an NVIDIA GeForce GTX 1060 graphics card with 6GB of GDDR5 VRAM. We really extracted each and every bit of its processing power so that we could train our network as much as we can. Even though the power wasn't that enough, the device was still used at its full extremity and gave us just the results that we can proudly present.

B. Software

To go as planned we require strong, powerful tools which will aid the processing and make things simpler without compromising the quality. To make the processes smoother we have used several software tools such as 'Anaconda' which is an open source Python data science platform, OpenCV [4] which is an open source Machine Learning and Computer Vision library which provides us with numerous programming functions. There is 'Keras' [5] which is also an open source Neural Network library in Python which helps us with the training. We also used TensorFlow-GPU [6] for showcasing our results in a more elaborate and illustrative form.

IV. PROCEDURE FOLLOWED

A. Dataset Collection

The main problem in any neural-network based application is the data. More the data, merrier it is. But for training a network to try to drive a car, we would need a lot of dashcam footage with exact steering angle at that exact point of time. Doing this at the level that we are working was difficult. Also, we cannot test a half-baked neural network on a real car because that would be dangerous. Now-A-Days, Open world games like Grand Theft Auto V, are made with such a detailing, that they almost simulate real world. This gives us a virtual world to play with and test our trained networks safely.

Methodology Used

In GTA V there exists an in-game AI that controls the people around us, drive cars, to make the world much more realistic. This AI can also drive cars. As this AI is specifically meant to control GTA V, it can drive cars in this world perfectly. Even

Fig (2): Some random screenshots from GTA V depicting the level of realism in the game.

though we are having a virtual world to simulate the real world-like conditions, it would be very tedious to drive a car in it for hours just to prepare a dataset; also driving the car manually would be very imperfect. So, we made use of a MOD for GTA V called 'DeepGTAV' [7]. This mod is made by 'Aitor Ruano' and is published publicly on GitHub. GTA V internal AI depends on scenarios in the world like driving mode, driving speed, vehicle, time in the world, weather, location in the world, and adapts right according to it as it is made specifically for that purpose. DeepGTAV helps us to set those parameters from our code directly, thus giving us a lot of control over the world scenarios. This give us immense control over all variables, recreate similar scenarios again and again but with slight variations each time. In our approach we are using the in-game AI to drive the car by itself to the waypoint set by us on the map. When we set a waypoint, the route to follow appears on

Fig (3): GTA V Screenshot

the map and mini-map in purple. The in-game AI follows this purple line to reach destination. This data of the in-game is captured and then stored. DeepGTAV also allows to extract the exact steering angle, throttle, brake,

location on the map, driving mode, speed, etc. from the game. This would help us to train our Neural Network to be trained in much better way. The dataset we are generating contains the current frame (frame currently being displayed on the screen), throttle, brake, steering, location and driving mode. This data is stored in a compressed '. pz' file to save space. Thus, by using the in-game AI we can generate huge amount of dataset without driving an actual car. The in-game AI communicates using sockets. The DeepGTAV communicates on port 8000. This socket can be used for communication of our code with the game GTA V itself thus bridging the gap between game and our program. Using this approach, we have generated 100GB of dataset which contains the parameters mentioned above. This dataset is given to the neural network for training.

B. Training

The Neural network model we used is the Inception V3 model. We are using this model in our code using the Keras API. Keras API provides various neural network models [5] like Inception V3, Xception, MobileNet, ResNet50, VGG16 and VGG19. It also provides us with pre-trained weights which we can use for fine-tuning and transfer learning. In our case we are using the Inception V3 architecture with no pre-trained weights. Previously we had tried to use transfer learning and fine tuning the Inception V3 weights which were trained for the ImageNet dataset [8] but it didn't give good results as the data we are trying to give to the neural network is of much different format than the ImageNet dataset. So, we changed to training our network with no pre-trained weights. The network was given the modified frame as input and the steering angle as the target. The steering angle present in the dataset ranges from -1 to 1 (floating point numbers). These values were scaled up to 0 to 999 (integers) which were later classified into 35 classes. These steering classes were then converted to one-hot arrays.

Fig (4): Pre-processed image given to neural network.

The modified frame contains the processed and enlarged Mini Map in top half part of the frame and the lower half was unchanged (contains the road). 4000 such frames were given to the neural network to fit to the target. This batch-size was decided according to the RAM consumption. For 16GB of RAM, the batch would consume about 90% of the RAM. If more frames were given, then it would result in OOM (Out-Of-Memory) error which would stop the program altogether and could potentially crash the system. The resolution of the input frame to neural network was also reduced to 299x299x3 (width=299, height=299, channels=3). This is to reduce the strain on the GPU, as it must process the frames and fit the model and weights to the target steering angle classes. The optimizer used is Stochastic Gradient Descent Optimizer with Nesterov momentum. After this, the network was first trained initially on 50GB dataset which contains about 60,000 frames with steering angles between 0 to 999. The training time taken for processing these for about 6 epochs is around 1 hour 35 minutes per epoch, thus in total, about 9 hours 40 minutes. The average accuracy obtained is around 26% (average accuracy of all batches) and the average loss obtained is 2.48. The results obtained after this training were not great. This is further elaborated in the Testing part. Later the network was tested on a 100GB dataset which originally contains 120,000 frames, but the neural network was given the frames only with steering angles between 0 to 470 and 530 to 999 which were about 52,000 frames. The training time taken for processing

this is about 1 hour 20 minutes per epoch thus in total it took 10 hours 50 minutes for 8 epochs of the same. The results obtained by these were noticeably much better as compared to previous but not up-to the mark. This is further elaborated in the Testing part. Later the steering angle which was scaled from -1 to 1 (floating point numbers) to 0 to 999 (integers) linearly, using the transformation

y = int (500 x + 500)

If y > 999, then y = 999

If y < 0, then y = 0

This transformation was changed slightly to train neural network to take sharper turns.
y = int (750 x + 500) If y > 999, then y = 999 If y < 0, then y = 0

Where,

x = steering angle in float (-1 to 1)

y = steering angle in integer (0 to 999)

The reason for this is explained in the Testing part. The network was trained in this data for about 6 epochs. The training time taken for processing these for about 3 epochs is about 4 hours 10 minutes. The results obtained by these were noticeably much better. It is explained in Testing part.

C. Testing

The testing of the network model and the weights was done using the 'DeepGTAV'[7] as it also provides the functionality to send the control signal from our program to the game using sockets. This information contains throttle, brake and steering. This information controls the car in the game. In our case, we are keeping throttle and brake constant and passing the output of our neural network to the game. Thus, our neural network is indirectly controlling the car in the game.

The initial training on 50GB did not produce good result. This was majorly contributed due to data imbalance. The number of frames with straight steering were much more than the frames with turns. This resulted in weights which were able to recognize the route and even steer according to it, it was even able to take smoother turns, but it was not able to take sharper turns like 90^o and 180^o turns as they comprised of very small percentage of dataset. To solve this, the model and weights were trained once again, but this time for only turn's dataset. This was done by only giving the frames to the model which had steering angle between 0 to 470 and 530 to 999. These values were chosen based on the statistical observations of the number of frames with those steering angles. These curved roads were taken by filtering them from a 100 GB dataset. It was trained for 10 epochs to balance the training. This did produce the result which we had expected. Now the car was able to take 90^o and 180^o turns, but on slower speed. As we aren't generating the brake data, the turns which the neural network expects that they will be sufficient aren't enough. This is because at faster speed, for same steering angle, the turning radius is more, and at slower speed, for same steering angle, the turning radius is less. So, our way to go around this issue was to amplify the steering angle predicted by our model. So, we changed the transformation equation of our steering classifier by increasing the conversion slope as mentioned above in the Training part. This was again performed using the 100 GB dataset for 10 epochs. The outputs obtained from this training were very good. The Network was able to take good decisions, but the issue of larger turning radius persisted in small amounts. This was solved by converting the linear scale from -1 to 1 of steering to a nonlinear sigmoid curve. This solved the problem by a great extent. These weights are performing very well as per our expectations and as per the amount of processing power we had. We did try to train these weights further, but the results were not good as it started overfitting on the dataset. The only solution to this is to create even more dataset, but due to time constraints we were unable to do so but will do it in the future. The current accuracy of the trained weights is around 0.35 which is good considering the number of epochs for which the network was trained.

Initially we decided to use a technique called transfer learning and fine tuning. This is a technique in which pre-trained weights are loaded and those weights are then trained upon with our dataset. The goal of this approach was that weights with good accuracy has already detected a lot of features with very high accuracy which requires huge amount of computing power and time. The weights we were trying to fine-tune were

trained on the ImageNet dataset [8] and had an exceptional accuracy of 93.9% top-5 accuracy [1]. This would have been a great starting point, but our dataset looks very different than the ImageNet dataset. Due to this, the pretrained Inception V3 [2] weights were not able to recognize the patterns in our input frame.

There are 2 major deep learning API used namely tflearn and Keras. We did try both API and found some significant performance differences in the two API. The tflearn API [9] was using a lot of CPU while using the GPU in short bursts thus not utilizing the computing power to its fullest and as the CPU usage is much higher, this contributed in higher CPU package temperatures due to which the CPU did start to thermal throttle.

Fig (5): TFlearn and Keras GPU consumption pattern.

V. FUTURE IMPROVEMENTS

Right now, the network is trained for about 100 GB of data. In future, when more computational ability and/or time is available, the network accuracy can be further improved, this would make the network much more confident about its decisions. The network can also be trained for special cases like coming back on road when the car has gone off-road, avoiding obstacles, detecting lanes, etc. by creating specific datasets which highlight such scenarios. This would make the network much more generalized.

Image processing can be used to detect lanes on road, detect traffic signals, detect other vehicles, etc. This would increase the confidence of the network. The image obtained can also be segmented to distinguish roads, footpaths, pedestrians, vehicles, etc. To perform these tasks simultaneously in real time, the computational power required would be much more than what we have right now. Multiple camera angles can be used to get much better idea about the surrounding of the car. With our current setup, right now, there is no way to get multiple camera angles from the GTA V game.

LIDAR/RADAR can be implemented to get a 3D depth map of the world around our vehicle so that we could get absolute distances between us and the surrounding objects. With our current setup, right now, there is no way to get that information from GTA V.

VI. ACKNOWLEDGMENT

We would like to thank Harrison Kinsley or popular as “Sentdex” on YouTube for inspiring us with your work and being a great motivation.

VII. REFERENCES

[1] Szegedy, Christian, et al. “Going Deeper with Convolutions.” 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, doi:10.1109/cvpr.2015.7298594.
[2] Szegedy, Christian, et al. “Rethinking the Inception Architecture for Computer Vision.”2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, doi:10.1109/cvpr.2016.308.
[3] Bottou, Léon; Bousquet, Olivier (2008). The Tradeoffs of Large Scale Learning. Advances in Neural Information Processing Systems. 20. pp. 161–168.
[4] I. Corporation, W. G. and I. , "OpenCV Library," [Online]. Available: https://opencv.org/.
[5] F. Chollet, "Keras: The Python Deep Learning library," [Online]. Available: https://keras.io/.
[6] "TensorFlow: An open-source machine learning framework for everyone.," Available: https://www.tensorflow.org/.
[7] A. Ruano, “aitorzip/DeepGTAV,” GitHub, 06-Nov-2017. [Online]. Available: https://github.com/aitorzip/DeepGTAV.
[8] "ImageNet," 2016 Stanford Vision Lab, [Online]. Available: http://www.image-net.org/.
[9] Damien, A. (n.d.). TFLearn: Deep learning library featuring a higher-level API for TensorFlow. Retrieved from http://tflearn.org/.