Programming an Autonomous Driving Car using Deep Neural Networks

Srushanth Baride
department of Physics and Astronomy
University of Nottingham
Nottingham, UK
ppxsb5@nottingham.ac.uk

In this paper I describe the life cycle, development procedure, techniques used, results and challenges faced to develop the Autonomous Driving Car using Deep Neural Networks. There are many steps taken to get the better results to the problem and some of them are pre-processing the images, try different transfer learning techniques and models, fine-tune the hyper parameters to get the better accuracy from the model. There is also a Neural Network which was developed from the ground up to train the model. Observations were made w.r.t the different models comparing to the model accuracy. The architectures of the transfer learning models were tweaked i.e., new layers were added and few layers were also removed in some cases.

INTRODUCTION

Autonomous cars in this days is a very controversial topic because of many reasons and since the Tesla has entered and disrupted the car manufacturing industry with electric cars and autonomous technology, there are many multi national companies started investing in to the electric car industry and trying to make their autonomous vehicles and there is a significant improvement has been made[1]. However there are many significant challenges which are needed to be addressed. There are many strtegies are being made, different technologies being used like LiDAR (Light Detection and Ranging), Computer Vision along with the Simulations of the scenarios to make this a fully autonomous and capable of very long distance trips without any requirement of human intervention. For the sake of this experiment, Deep Neural Networks has been used.

VGG NET

VGG Net is a type of CNN (Convolutional Neural Network)[2] named after the Visual Geometry Group from Oxford. The number after the network represents the number of layers the network like listed below.

1. VGG-11
2. VGG-11 (LRN (Local Response Normalization))
3. VGG-13
4. VGG-16 (conv1)
5. VGG-16
6. VGG-19

VGG has a very simple architecture which is built using CNN, 3X3 filters with stride and a the padding size of 1 and a max pooling size of 2 with stride 2, pooling layers and a fully connected layer.

VGG is one of the most used image-recognition architectures however, smaller network architectures are preferred. The FIG 1 is showing the architecture of VGG16 and has 138 milliion parameters and is one of the largest models. The architecture is showing as the image goes deeper into the network, the size of the image is reduced drastically because of the increase in the number of filters.

RESNET

ResNet is short for Residual Network. A team at Microsoft has developed this network. This is to improve the training of the network which are more deeper than the previous models. As the network grows deeper due the stacking of multiple layers, the accuracy of the model will start to decrease after a certain point, this is called as "degradation problem". ResNet uses "residual mapping" technique to tackle the issue.

In ResNet, there are fewer filters and is not very complex as VGG net. A shortcut connection for identity mapping, with extra zero entries padded for increasing dimensions is made to make it into counterpart residual version.

Below are the variables which can be tweaked to change the residual blocks in the network.

1. Convolution type
2. Convolutions per block
3. Width of residual blocks
4. Dropout

INCEPTION V3

Inception[4] is a Deep Convolutional Neural Network architecture that can perform Classification and detec-

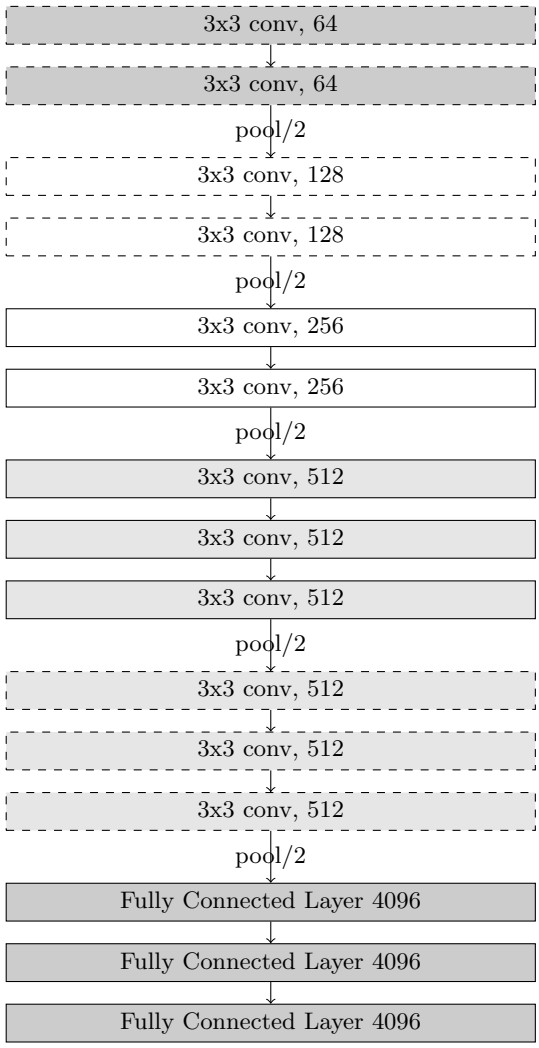

FIG. 1. VGG16 Architecture[3]

tion. The architectural decisions were based on the Hebbian principle and the intuition of multi-scale processing and this network has been modified to be deeper and wider without affecting the amount of computational requirements. If any changes[5] need to be done to the existing network, care needs to be taken that the computational advantages are not lost and therefore the use of Inception network for different use-case may turnout to be problem due to the uncertainty of the new network. Even though it's tricky to the Inception network but there are many network optimizing techniques in place to loosen up the constraints.The techniques include factorized convolutions, regularization, dimension reduction, and parallelized computations. The number after V in Inception V3 stands for the version of the model. This model returns a image classification model, optionally loaded with weights pre-trained on ImageNet.

The accuracy obtained by the Inception V3 model on the ImageNet dataset is greater then 78.1%[6].

Architecture:– The Inception V3 architecture is built sequentially as below.

1. Factorized Convolutions: Reduces the number of parameters which helps in reducing the computational efficiency.
2. Smaller convolutions: This replaces larger convolutions with smaller convolutions which reduces the number of parameters and in-turn reducing the time required for training the model.
3. Asymmetric convolutions: Instead of 3×3 filter, 3×1 and 1×3 filter is used to make the network more diverse.
4. Auxiliary classifier: A small CNN which is added between the layers at the time of training and loss occurred to the addition due to the network is added to the main network.
5. Grid size reduction: This is mostly done using pooling method.

TRANSFER LEARNING

Transfer Learning[7] is a research problem in the field of Machine Learning and Artificial Intelligence which focuses on storing the gained knowledge trying to solve different problems and using the same gained knowledge to solve the similar problems. This technology is gaining a rapid public fame due to it's high results.

In Computer Vision, neural networks are majorly focused on extracting the features from the images like edges in the first layers, forms in the middle layer and the problem related operations in the later layers. The first and the middle layers are used in the transfer learning and the remaining layers are re-trained[8] as per the problem which is being worked on.

The Transfer Learning can be used[8] when there is no enough data to be trained but you have a model which is already trained on the similar problem and giving the better results. The weights from the similar pre-trained model can be re-used to train on new data which is trying to solve a new problem. In this process the layers which are responsible for feature extraction will be unchanged but the layers which are responsible for solving the problem will be retrained, because of this, the parameters which are required to train will be drastically reduced in number and the network will be trained even faster. Because of this feature, even very large networks also can be trained faster along with good results.

IMPLEMENTATION PROCEDURE

There are multiple approaches planned to solve the problem and are listed below.

1. Lane Detection
2. Object Detection
   (a) Tree
   (b) Cube
   (c) Human
   (d) Directions
   (e) Red and Green signals
3. Model for Angle and Speed

The initial approach was to build the Object detection model to detect the pre-defined objects like trees, human, cube, etc and feed that to the angle and speed model to get the required results. Along with this a lane detection was also planned to get the only region of interest by masking the rest of the area. By doing this, even if the human is standing at the side of the road, it will be completely ignored for the speed model.

Constraints:–

1. The available compute is very low.
2. Cannot run huge models
3. Cannot modify the hardware to install new sensor components

Hardware:– The most important component is Raspberry Pi[9] 4B with a decent 1.8 GHz processor. Raspberry Pi is a series of very small single board computers (SBCs) which is manily used of conducting PoCs (Proof of Concepts) and many DIY (Do It Yourself) projects and is developed in the United Kingdom by the Raspberry Pi Foundation in association with Broadcom. The 2019 version of Raspberry Pi 4 comes with updates for better load and run-time. Every computation which is performed for this project is done on this piece of hardware.

Approach:–

1. Load all the required libraries
2. Load the training data, file size from the defined directory and create a data-frame of it
3. Load the csv file containing the speed and angle of the car mapped to the loaded image id
4. Join the training data data-frame and loaded csv with join operation
5. Remove the sample instances with file size less than 0
6. Check the distribution of speed and angle for any abnormalities
7. Data Augmentation
8. Split the data into train and test dataset's
9. Pre-process the image's
   (a) Read the image
   (b) Convert the image from RGB (Red, Green and Blue) to YUV (luma, B-Y luma and R-Y luma)
   (c) Blur the image using GaussianBlur
   (d) Resize the image based on the size requirement of the Neural Network
   (e) Normalize the complete image with 255 since it's the standard max value of the selected color space
10. Build the Neural Network
    (a) From Scratch
    (b) VGGNet
    (c) InceptionV3
    (d) ResNet50
11. Define the model output directory for saving the final model
12. Define callback functions
    (a) To check the status of the training
    (b) To save the model once the condition is met
13. Fit the model with defined
    (a) Training data
    (b) Validation data
    (c) Batch size
    (d) Number of epochs
    (e) Steps per epoch
    (f) Callbacks
14. Plot the meta data to check the model accuracy
15. Validate the model using test data
16. Serve the model

As mentioned in the above approach, there were many steps taken and experiments performed to conclude on which model is to be used, how hyper-parameters needs to be fine-tuned, what learning rate should be used, what should be the batch size and what should the number of steps per epoch.

Initially, the challenge was to get a decent accuracy from the model and build the rest on top of that. Building the model from ground-up was a challenge because of the number of decisions needed to make like

- What filters needs to be used
- What should be size of the each filter
- How many layers are needed
- When the dropout needed to be added
- How much of the neurons needed to be dropped out

The list of the decisions needed to make went on as the project progressed.

PRE-PROCESSING AND AUGMENTATION

Pre-processing:– Pre-processing are the steps taken to re-format the data (in this case they are images) before they are used to build, train and use the models. Pre-processing is one of the important steps while solving any data problems and they include but not limited to adding random noise, change to gray scale, normalise.

Image pre-processing puts a significant effect on how much time it takes to train the model and how fast the inference time will be once the model is ready. Re-sizing of the image will put a great impact on how long it will take to train the model. If the image size if very large, it's going to take a long time to train the model but if the image is reduced in size at the time of pre-processing, the amount of time needed to train the model will be drastically reduced. The same is applies for the inference time as well. If the size of the image is very large, the time required to infer will be high but if the image is re-sized before inference, the inference till will be drastically reduced. This is not limited to the size of the image but many others can have the same affect on this. Some of the pre-processing techniques are listed below.

1. Re-size
2. Random flip
3. Add noise
4. Change the color space
5. Change the image orientation
6. Normalise
7. Crop

In this project, the following steps are taken to pre-process the complete dataset and same steps are applied to the image stream before performing the inference and they are listed below.

1. Crop the image horizontally to the half of its height to focus on the required target and eliminate any un-necessary space
2. Convert the image color space from RGB to YUV as Nvidia model said it is best to use YUV color space
3. Apply Gaussian Blur
4. Re-size the image to a (x, y) size based on the type of model chosen for the training and the input shape requirements
5. normalizing the image

Augmentation:– Image augmentation is one of the important tasks to perform. It is performed on the data to create different versions of the same data which will act like a new dataset in return increasing the size of the entire dataset. Caution needed to be taken while performing data augmentation as few augmentations can also add bad data which is not required and in-turn reduce the model accuracy. In this project, flip is one of such operations which has been made. While performing the flip, the steering angle also needed to be changed i.e., $180^o - steeringangle$ to get the correct angle when the image is flipped.

Some of the Image augmentation steps taken are listed below.

1. Pan
2. Zoom
3. Blur
4. Brightness
5. Flip

RESULTS

There are many Deep Learning models which was built from ground up and using the technique i.e., transfer learning.

Below is the table showing the final result for the VGG16 model which is responsible for angle.

Epochs	Loss	Validation Loss
60	0.0073	0.0081

Below is the graph showing the training and validation loss trend while training the model for angle.

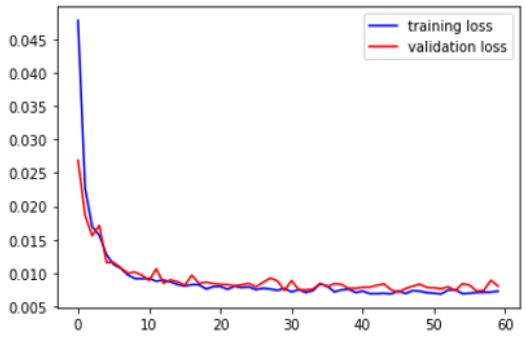

FIG. 2. Training vs Validation loss for Angle

From the Figure 2 above, we can see that the training loss dropped exponentially from 0.045 to 0.010 in the first 10 epochs and gradually reduced to 0.004 for the rest i.e., till the 60^{th} epoch.

Below is the table showing the final result for the VGG16 model which is responsible for speed.

Epochs	Loss	Validation Loss	Accuracy	Validation Accuracy
30	0.0788	0.1133	0.9704	0.9539

Below is the graph showing the training and validation loss trend while training the model for speed.

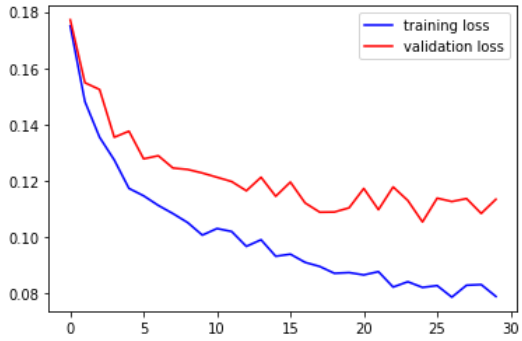

FIG. 3. Training vs Validation loss for Speed

Below is the table showing all the metrics from experiments done on Densenet121 model with SGD optimiser.

LR	Train Batch Size	Val Batch Size	Epochs	Loss	Val Loss
0.01	64	64	30	0.0067	0.0078
0.01	64	64	20	0.0071	0.0077
0.01	32	32	30	0.0073	0.0078
0.01	32	32	20	0.0071	0.0078
0.001	64	64	20	0.0088	0.0098
0.001	32	32	25	0.0081	0.0089
0.001	32	32	20	0.0078	0.0084
0.0001	64	32	30	0.0117	0.0109
0.0001	64	32	20	0.0134	0.0139
0.0001	32	32	30	0.0087	0.0096
0.0001	32	32	25	0.0094	0.0097
0.0001	32	32	20	0.0112	0.0117
0.00001	64	32	20	0.0221	0.02259
0.00001	32	32	20	0.0174	0.01698
0.00001	32	32	25	0.0182	0.0191
0.000001	32	32	20	0.0345	0.035

Below is the table showing all the metrics from experiments done on Densenet169 model with SGD optimiser.

LR	Train Batch Size	Val Batch Size	Epochs	Loss	Val Loss
0.01	32	32	20	0.007	0.0089
0.01	16	16	20	0.0078	0.0094
0.01	64	64	20	0.0066	0.0078
0.01	128	128	20	0.0063	0.0074
0.001	32	32	20	0.0068	0.0079

From the Figure 3 above, we can see that the training loss and the validation loss are very close but not merging. This is the case because the scale at which the graph is plotted. In reality, the acceptance criteria for the model producing the results as shown in the figure 3 will be very less if that loss was in the multiples of thousands but in this case, the loss is in fractions and the produced result is very much acceptable.

Based on the number of experiments performed, the general conclusion reached is that the heavier models will perform better up to certain extent but after a certain number of layers, the accuracy of the model will keep degrading and using the technique known as transfer learning will save a lot of time and computational power for doing any similar type of tasks which the original model was built on. The results from the transfer learning model is also very high.

[1] M. Martínez-Díaz and F. Soriguera, Transportation Research Procedia **33**, 275 (2018), xIII Conference on Transport Engineering, CIT2018.
[2] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, Neural Computation **1**, 541 (1989).
[3] R. Shanmugamani and S. Moore, *Deep Learning for Computer Vision: Expert Techniques to Train Advanced Neural Networks Using TensorFlow and Keras* (Packt Publishing, 2018) p. 59.
[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015) pp. 1–9.
[5] V. Kurama, "A Guide to ResNet, Inception v3, and SqueezeNet," (2021).
[6] Google, "Advanced Guide to Inception v3 — Cloud TPU — ," (2022).
[7] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, Proceedings of the IEEE **PP**, 1 (2020).
[8] P. Sharma, "Transfer Learning — Understanding Transfer Learning for Deep Learning," (2021).
[9] Wikipedia contributors, "Raspberry Pi," (2022).