Programming an Autonomous Driving Car

Divyansh Oze

University of Nottingham

ppxdo1@nottingham.ac.uk

May 20, 2022

Abstract

This report is part of a Machine Learning project explaining the programming of an autonomous driving car. We have experimented with developing an autonomous-driving model using several approaches such as end-to-end learning, transfer-learning and object detection to predict steering commands like angle and speed based on the images of the track. The system learns to drive along the track, following the lane markings and stopping at obstacles on the road. It learns to stop and go at traffic lights and turn left or right according to the signals on the road. The report concludes with further advancements in the approaches used during the project.

Keywords: Autonomous Driving, Machine Learning, Convolutional Neural Networks

I. Introduction

The Applications of Artificial Intelligence and Machine Learning are progressively widespread and permeate through fields, more from the last few decades, in autonomous driving. Many technological giants such as Waymo and Uber have led the use of Artificial Intelligence and Machine Learning [I Sonata,2017] to replace traditional systems like GPS and radar to detect the surroundings.

With the implementation of Artificial Intelligence and Machine Learning approaches, tremendous success has been made in regions of image classification and recognition tasks. Convolutional Neural Networks (CNNs) have predominant use in computer vision tasks for Autonomous Driving. CNNs have also shown outstanding performance on semantic segmentation tasks [Çağrı Kaymak (2019)], which can be used to output every pixel of an image targetting an object class. The object classes for a driving environment could be "vehicle, pedestrian, traffic sign, signboard', basically helping the cars to identify driveable portions from the image seen of the road.

In this paper, the model was trained on several approaches to implementing autonomous driving in a live track with different patterns and objects placed on and off the track, with varying difficulties in understanding the surroundings to make a decision.

II. Related Work

There is considerable research on the autonomous driving car using deep learning approaches. The different levels of automated driving can be learnt from [Maurya Vijayaramachandran, (2020)], which covers the spectrum moving from one end of the human monitoring to the automated system monitoring the driving environment.

The generic obstacle and lane detection (GOLD) system [M. Bertozzi, (1998)] combined lane-position tracking with obstacle detection for self-driving freeway driving. It introduced a function to find lane markings in an inverse-perspective image of a road based on differences in brightness between a pixel and its neighbours to the left and right. This technique was preferred for a significant amount of time as the approach focused on getting geometric features from the track to extract features conveniently from the background.

Over the past decades, many types of research have managed to revolutionise pattern recognition [Mariusz Bojarski(2016)]. Earlier, the collection from hand-crafted feature extraction made pattern recognition tasks possible to run predictions. The invention of CNN brought in automated feature learning for such image recognition tasks to understand and inherit essential features from the digital images without human intervention. The learning capacity of the convolutional neural network controlled by varying its depth and breadth is another reason to implement such digital problems with CNN's.

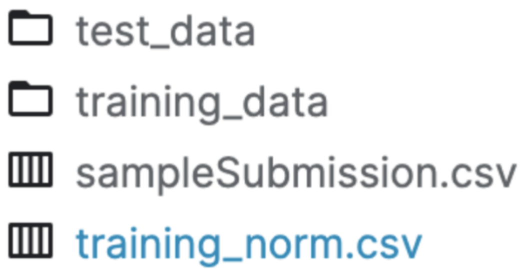
CNN overpowers the standard feedforward neural networks as they have fewer connections and parameters [Maurya Vijayaramachandran, (2020)], enabling them to get trained faster and more efficiently to save computational time. The approach for CNN is incredibly robust in image recognition tasks as the convolutional operations capture the 2D nature of an image. During the time of CNN, the different learnings introduced within the learning to approach image-recognition tasks were based on unique architectures and extracting the features given the task. These included the learning as mentioned in [[Jianan Li, (2017)], [Maurya Vijayaramachandran, (2020)], [Joseph Redmon (2016)]]

The research in [Jianan Li, (2017)] projects out the usage of Fast-RCNN to detect pedestrians on the track. The paper displays superior performance and computational efficiency by detecting the general objects spotted on the road. A new convolutional network, YOLO (You Only Look Once) [Joseph Redmon (2016)], was introduced for object detection. YOLO works in real-time, which, unlike the traditional sliding window and region proposal-based techniques, sees the entire image during training and testing, i.e. implicitly encoding contextual information about the total classes and their appearance. Due to its inability to see the larger context in images, Fast R-CNN often mistakes patches in images for objects. The newly proposed YOLO makes fewer mistakes in detecting such objects than Fast R-CNN. [Maurya Vijayaramachandran, (2020)] encompasses how YOLO works out for self-driving cars by using RADAR, Ultrasonic Sensors and LiDAR to help recognise the distance between the ego-vehicle and surrounding objects in the background. Except for the vision-based module for the pedestrian module, researchers adopted 3D LiDAR sensors to detect and track pedestrians; as shown in [Jian Dou, (2017)], the geometrical clues generated by the sensors passed through a Support Vector Machine (SVM) for class information.

Despite CNN's having all the attractive qualities and relative efficiency of their local architecture, CNNs have been expensive to apply in enormous cases with high-resolution images, as discussed in [Maurya Vijayaramachandran, (2020)]. Fortunately, when paired with a highly-optimised implementation of 2D convolution, the development of GPUs enables even to train the seemingly large CNNs. A dataset like ImageNet that we would use for our pre-trained model contains enough labelled images to train the models without overfitting them.

III. Data Description

The Dataset retrieved from [MLiS-2 Data] consisted of all the files as shown in Figure[1] A total of 14.8k images were provided for the project. 13.8k were for the training set and the remaining for testing the model.



test_data
training_data
sampleSubmission.csv
training_norm.csv

Figure 1: *Dataset retrieved from Kaggle*

Images from the Dataset were collected from driving over the track. Objects like "pedestrians" and "traffic signals" were placed on and off-track to derive decisions from the vehicle. Figure[2] shows an example of an object placed on the track while the vehicle is driving over the track.

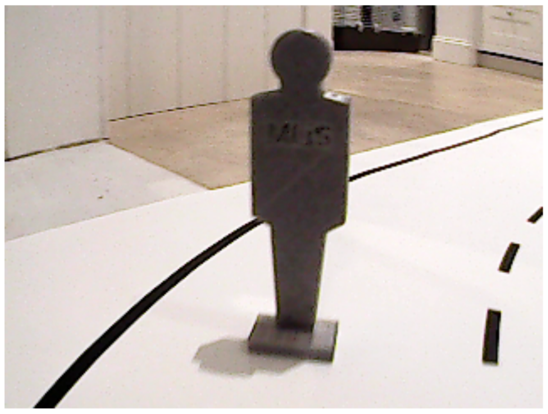
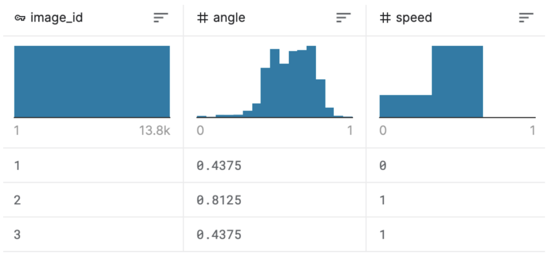


Figure 2: *Pedestrian standing on the driving-track*

The training_norm.csv file contained the ground truth labels to the training set images. The structure of the training_norm.csv file is shown in Figure[3]. The file contained knowledge of the car's environmental state (images) on each action(steering commands, i.e. angle, speed) taken by the vehicle.



image_id	angle	speed
1 – 13.8k	0 – 1	0 – 1
1	0.4375	0
2	0.8125	1
3	0.4375	1

Figure 3: *Structure of the training_norm.csv file*

IV. Methodology

A technique inspired by the biological visual perception, i.e. CNN, has been in all of the approaches for the task of autonomous driving. It contains three basic operations: convolution, pooling and fully connected. The convolution process transforms the given input image into information beneficial [Yann LeCun,(1995)] for the network in extracting features.

Following are the models that we have developed to approach autonomous driving:

- End-to-End Model
- Transfer Learning Model
- Object Detection Model

i. End-to-End Model

The End-to-End Model is greatly inspired by Nvidia Model [Mariusz Bojarski(2016)], so the name for the model is used interchangeably. The network used for this model consisted of 9 layers, 5 convolutional layers, and 3 fully-connected layers, as shown in the architecture [5]. As advised for the Nvidia Model, the input image splits into YUV planes to pass into the network. The first layer in the network performs Image Normalisation to allow the normalisation scheme to be altered with the architecture and accelerated via GPU processing.

Feature Extraction is performed by the convolution layers, finalised after analytical experimentation that varied layer configurations. The convolution layers use strided-convolutions in the first three layers with a stride of (2x2) and a size kernel (5x5). The last two convolution layers used non-strided convolutions with a size kernel (5x5).

Following the Convolutional Layers, three fully connected layers are added for further computation, leading to the vehicle's speed and steering angle output.

i.1 Data Preprocessing

The images and respective image indexes [1][3] are combined as a component for input to the CNN. The steering commands for the vehicle, speed and angle were stored as labels to predict given the images. The preparation works the same for both the train and test sets.

The OpenCV [InceptionResNetv2] library was used to modify the images with the in-built functions like GuassianBlur, RGB2YUV, and Resizing, based on which an example is shown in Figure[4] based on the requirement.

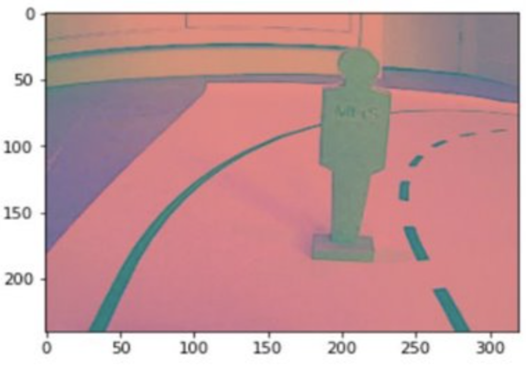


Figure 4: *A pre-processed image in YUV format*

i.2 Training

In training the end-to-end model, the training images are passed onto the model to fit for 10 epochs with a batch_size of 100 with Adam Optimizer for minimising the loss.

i.3 Hyper-Parameter Tuning

Before running the model predictions on the test data, it is first determined on the validation set. The validation dataset is run for 10 epochs with a batch_size of 100.

The model was made to undergo hyper-parameter tuning in order to stabilise the error rates and make it generalise better for the test data:

Activation Functions: The model uses ELU (Exponential Linear Unit) as an activation function for each layer. The activation function provides the nonlinear expression ability for the neural network to enable the model to fit better with the results and improve the accuracy. In contrast to ReLUs (Rectified Linear Units), it has negative values, allowing them to push the mean unit activations closer to zero.

Dropouts: Dropouts are added to the network to prevent the model from overfitting on the training data.[Shaofeng Cai (2020)] Without dropout, the training tends to stagnate quickly, and the test error rate increases and fluctuates dramatically before the first drop in the learning rate. We use dropouts in one of the convolutional layers and the other in the fully connected layers. While using dropouts, the training for the model is more stable, enabling the network to continue improving for lower errors.

BatchNormalization: Batch Normalisation is a technique done between layers of the neural network instead of just normalising the raw input data. It is implemented using mini-batches to speed up training and make learning easier for the model. 4 Batch Norm Layers are added between the Convolution Layers and between Dense layers to ensure updated parameters during back-propagation do not affect the overall distribution for our model.

Regularizers: To address the famous bias-variance trade-off, L2 Regularizers, i.e. commonly known as weight decay, are put on convolution and dense layers to limit our model from over-fitting on the training data. Regularizers help modifies our loss function by penalising specific large values of weights that the model learns. The model avoids over-fitting given the less complex network.

Loss Optimizer: In order to optimise the loss function that measures the errors for our model with the given approach, optimisation functions are used to reduce the errors in predictions. As shown in [Diederik P. Kingma, (2017)], for the tremendous results in deep learning networks, Adam was used in implementing the approach.

Learning Rate and Batch_Size: We wanted to balance the trade-off between using learning rates and Batch_Size for the model approach by choosing the correct learning rate if the size of the batch makes the training process noisy. A low learning rate of 0.001 has been taken for a batch size of 100.

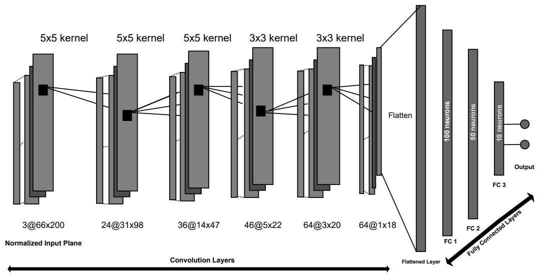


Figure 5: *Nvidia Model Architecture*

i.4 Evaluation

The test scores were recorded from Kaggle after submitting a .csv file consisting of image indexes, angle and speed predictions for the vehicle.

ii. Transfer Learning Approach

Transfer Learning in Machine Learning is a technique whereby a model is trained and developed for one task and is then re-used on a second related task. It refers to the situation whereby what has been learnt in one setting is exploited to improve optimisation in another setting [Mahbub Hussain (2018)]. The pre-trained model is generally used for transfer learning if the new dataset on the task is smaller than the one it has been trained on before.

The two models combined to develop this second model are given as follows:

- **InceptionResNetv2** : The InceptionResNetv2 is used to predict the speed as steering commands for the autonomous-vehicle.
- **Nvidia Model**: The Nvidia Model is used to predict the angle at which the steering should be rotated for the autonomous vehicle.

ii.1 InceptionResNetv2

Inception-ResNet-v2 is another convolutional neural network pre-trained on the ImageNet database, i.e. containing more than a million images. The inception network is 164 layers deep, and about 1000 object categories can be classified on images using the InceptionNet.

The model uses the Inception-ResNet-v2 since it has learned essential rich feature representations for various images, and the tuned weights can be beneficial for helping this model know better.

ii.2 Data Preprocessing

The pre-processing performed on the second model is about the same as in Nvidia Model using the OpenCV library. The change made in partitioning the data columns for speed and angle is shown in Figure[6].

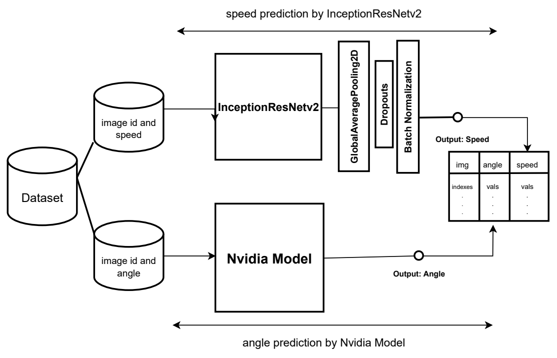


Figure 6: *Architecture for the Transfer Learning Model*

ii.3 Training

After getting the speed and angle column values loaded for the InceptionResNetv2 and Nvidia model, The independent models generate out predictions for steering commands, i.e. speed and angle for the vehicle based on the input image shown.

ii.4 Hyper-Parameter Tuning

The Nvidia Model shown in Figure[6] represents the optimised pre-trained model from the first example. Global average pooling, Dropouts and Batch Normalization are added to the results generated from the InceptionNetv2 to avoid letting the model over-fit on training data.
Evaluation The column predictions for speed and angle are collected from the Inception and Nvidia Model to combine and store in a .CSV file with the respective image indexes to be called at Kaggle for a score of test error. The test_loss, i.e. combined test_loss after combining both the model's predictions, is shown below in Table [2].

iii. Object Detection

The object detection model was aimed to work on labelled images, with information about the objects in the given input, the coordinate and size details about bounding boxes applied to detect the objects. We used the MobileNetSSD2 to perform the object detection task, though we could not obtain the desired results.

The labelled images would help the model identify the objects to make better decisions accordingly in the driving environment. The labelling for images stored in the training and testing set was done manually by using LabelImg [tzutalin], i.e. tool to put a graphical annotation on images.

We managed to label a sufficient number of images for all objects seen in the training dataset. (1. Pedestrian 2. Red Light 3. Green Light 4. Right Signal 5. Left Signal 6. Box 7. Tree 8. Car). The following files are stored in a .XML format with information about the images.

V. Results

i. Nvidia Model

The Nvidia Model trained for a total of 10 and 20 epochs, using a batch_size of 100 and learning rate of 0.001 is shown in Figure [7] and Figure [8]. Losses obtained for the Nvidia Model after Hyper-Parameter Tuning is shown in Table [1]

epochs	train_loss	val_loss	test_loss
10	0.0053	0.0140	0.0321
20	0.0039	0.0153	0.0337

Table 1: *Hyper-parameter tuning results*

ii. Inception Model

The Inception Model used two models to predict the values for speed and angle, as described in Figure [6], to generate results for training and validation. The generated results combined with having a test loss

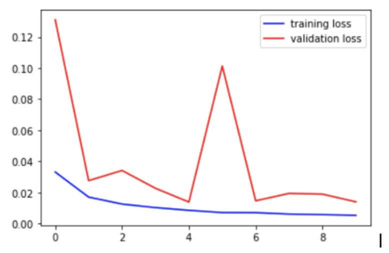


Figure 7: *Nvidia Model trained on 10 epochs.*

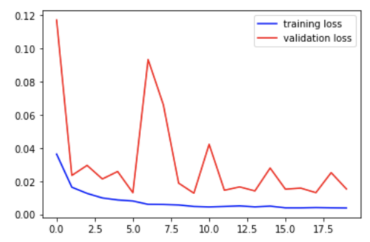


Figure 8: *Nvidia Model trained on 20 epochs*

from Kaggle as shown in Table[2].

Model	train_loss	val_loss	test_loss
Nvidia Model	0.0033	0.0076	0.072
InceptionResNetv2	0.0712	0.0723	0.072

Table 2: *Resultant losses from the Transfer-Learning Model*

iii. Kaggle Scores

The resultant .csv file submitted on Kaggle with the format [3] had the following scores generated based on the test errors, as shown in Table [3]

Model	Public	Private
End-to-End Learning	0.0327	0.0397
End-to-End Learning(v2)	0.0294	0.0334
Transfer-Learning	0.0771	0.0724
Transfer-Learning(v2)	0.0753	0.0701

Table 3: *Kaggle Leaderboard scores for Models*

iv. Live Testing

After seeing a trained network demonstrating a great performance in predicting the steering commands for the vehicle and an inference time of 100ms-120ms, the network is loaded on the Raspberry Pi (RPi) 4 to take out on a live test. We ran our live test using the Nvidia Model, which gave out a reasonably good inference time and could decide for the objects placed on the track, i.e. stop or go. It failed to identify the 'Green Light' on Traffic Signal to which we speculate that the model predicted it to be a standard object on the track. The model run on the tracks managed to get a score of 14 for live testing as per the marking criteria.

VI. Conclusion and Future Prospects

- The Nvidia Model provided great accuracy [3] considering it had a basic architecture and wasn't that deep of a neural network. The BatchNormalization layers kept in between the Convolutional and Dense Layers helped train the model faster and more smoothly. It provided us with a test score of 0.029 after adding Dropouts, Regularizers indicating that it was generalising well for the data it had not seen before. There were attempts made to train the Nvidia Model for more than ten epochs, but the model tends to over-fit on the training data and fails to provide any better loss performance for running it on twenty epochs, as shown in table [1].

- The Inception Model should have performed much better considering InceptionResNetv2 (a much deeper network) to predict speed in this second network. However, there is a possibility that the model needed more epochs to run to be trained better for the task or that we did something wrong in the conversion of files to predict the test scores for this model. It is also important to note that the Nvidia model performed extraordinarily well in predicting just the angles for the model given the input images.

– The object detection model could make use of Regional Proposal Networks [Shaoqing Ren (2015)] in order to tackle cases where spotting an object on and off the track could be problematic

References

[Akhil Agnihotri, 2019] Akhil Agnihotri, Kriti Rajesh Bapnad ,Prathamesh Saraf (2019). A Convolutional Neural Network Approach Towards Self-Driving Cars, *2019 IEEE 16th India Council International Conference (INDICON)*.

[Mariusz Bojarski(2016)] Davide Del Testa,Daniel Dworakowski,Bernhard Firner(2016). End to End Learning for Self-Driving Cars, *IJANA*.

[I Sonata,2017] Y Heryadi, L Lukas, A Wibowo, (2020). Autonomous car using CNN deep learning algorithm *Annual Conference on Science and Technology (ANCOSET 2020)*.

[Jianan Li, (2017)] Jianan Li, Xiaodan Liang, Shengmei Shen, Tingfa Xu (2017). Scale-Aware Fast R-CNN for Pedestrian Detection, *IEEE Transactions on Multimedia (IEEE)*.

[Syed OwaisAli Chishti,(2018)] Syed OwaisAli Chishti, Sana Riaz, Muhammad BilalZaib, Mohammad Nauman(2018). Self-Driving Cars Using CNN and Q-Learning *IEEE*.

[Maurya Vijayaramachandran, (2020)] Maurya Vijayaramachandran, Akilesh 2020). Use of CNN(YOLO) in Self Driving vehicles *IOSR Journal of Computer Engineering (IOSR-JCE)*.

[Jian Dou, (2017)] Jian Dou, Jianwu Fang, Tao Li, Jianru Xue, 2017). Boosting CNN-Based Pedestrian Detection via 3D LiDAR Fusion in Autonomous Driving *LNIP,volume 10667(LNIP)*.

[Yann LeCun,(1995)] Yann LeCun, Yoshua Bengio 1995). Convolutional Networks for Images, Speech and Time-Series *The Handbook of Brain Theory and Neural Networks*.

[Maurya Vijayaramachandran, (2020)] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton 2012). ImageNet Classification with Deep Convolutional Neural Networks *Advances in Neural Information Processing Systems 25 (NIPS 2012)*.

[M. Bertozzi, (1998)] M. Bertozzi,A. Broggi 1998.). GOLD: a parallel real-time stereo vision system for generic obstacle and lane detection *IEEE Transactions on Image Processing (IEEE)*.

[Joseph Redmon (2016)] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, 2016). You Only Look Once: Unified, Real-Time Object Detection *arXiv:1506.02640*.

[Shaofeng Cai (2020)] Yao Shu, Wei Wang, Gang Chen 2020). Efficient and Effective Dropout for Deep

Convolutional Neural Networks *IEEE*.

[Diederik P. Kingma, (2017)] Diederik P. KingmaJimmy,Lei Ba 2017). ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION *IEEE*.

[Çağrı Kaymak (2019)] Çağrı Kaymak, Ayşegül Uçar (2019)). Semantic Image Segmentation for Autonomous Driving Using Fully Convolutional Networks *International Artificial Intelligence and Data Processing Symposium (IDAP)*.

[Shaoqing Ren (2015)] Shaoqing Ren, Kaiming He,Ross Girshick, Jian Sun(2015)). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks *arXiv:1506.01497*.

[tzutalin] tzutalin). LabelImg, graphical image annotation tool. *https://github.com/tzutalin/labelImg*.

[MLiS-2 Data] . Machine Learning in Science II Autonomous driving project. *https://www.kaggle.com/c/machine-learning-in-science-2022/data*

[OpenCV] . Computer program, *https://opencv.org/*.

[Mahbub Hussain (2018)] Mahbub Hussain, Jordan J. Bird, Diego R. Faria (2018)). A Study on CNN Transfer Learning for Image Classification, *Conference: UKCI 2018: 18th Annual UK Workshop on Computational Intelligence*.

[InceptionResNetv2] . InceptionResNetV2 function. *https://keras.io/api/applications/inceptionresnetv2/* .