Research on Autonomous Driving Car Using Deep Learning

HoHim Lee[1, *]

[1] *School of Physics and Astronomy, University of Nottingham, Nottingham, NG7 2RD, UK*

(Dated: October 21, 2023)

In this project, we discover the possibilities of using various machine-learning models including self-trained models and pre-trained models to simulate self-driving capabilities on the Picar-V from SunFounder [1].

I. INTRODUCTION

In recent decades, self-driving cars, also known as autonomous vehicles, have been one of the most popular research areas for machine learning and computer vision scientists. Self-driving cars represent vehicles that have the capability to control themselves without the need for human intervention, which includes navigating themselves in a variety of environments and reacting to the actions of other road users. The biggest concern about self-driving cars is the safety aspect, the constant development of self-driving technologies in the past few years has greatly improved the safety and energy-saving performances of these self-driving vehicles, recent studies also suggest that self-driving cars are safer than human drivers and can reduce the number of accidents caused by human errors [2]. There are three main components for autonomous vehicles, environmental perception, decision making and motion control [3]. Each of them contributes to the performance of the autonomous vehicle. Currently, the main idea to develop autonomous vehicles is to use various machine-learning techniques, which include finite-state machines [4], decision trees [5], deep learning [6], reinforcement learning and inverse-reinforcement learning [7].

Driving automation systems are categorized into 5 different levels defined by the Society of Automotive Engineers (SAE), the higher the level, the less human intervention will be needed for the vehicle to navigate themselves [8]. Currently, most of the production cars with sefl-driving systems are at level 2, such as Tesla AutoPilot and General Motors Supercruise [9]. Both of these provide features like automatic emergency braking, lane centring and adaptive cruise control, but constant attention of the human driver is needed in order to maintain safety.

The goal of this project is to investigate the implementation of deep learning in driving automation systems. There are two objectives for this goal, the first being to develop a deep-learning model that can predict the appropriate speed and steering angle based on an image, and the second is to perform self-driving on a model car around a set of realistic test circuits. The model car used in this project will be the Picar-V from SunFounder [1], it is based on a Raspberry Pi 4B and included a camera to achieve real-time automatic navigation.

The dataset used in this study consists of 13,798 images along with the target car response (speed and steering angle), these images were taken on the car itself on the test circuits under different environments.

The rest of this report will outline the related works, methodologies used, results and findings. We will then finish with a further work section followed by a conclusion of our report.

II. PREVIOUS WORKS

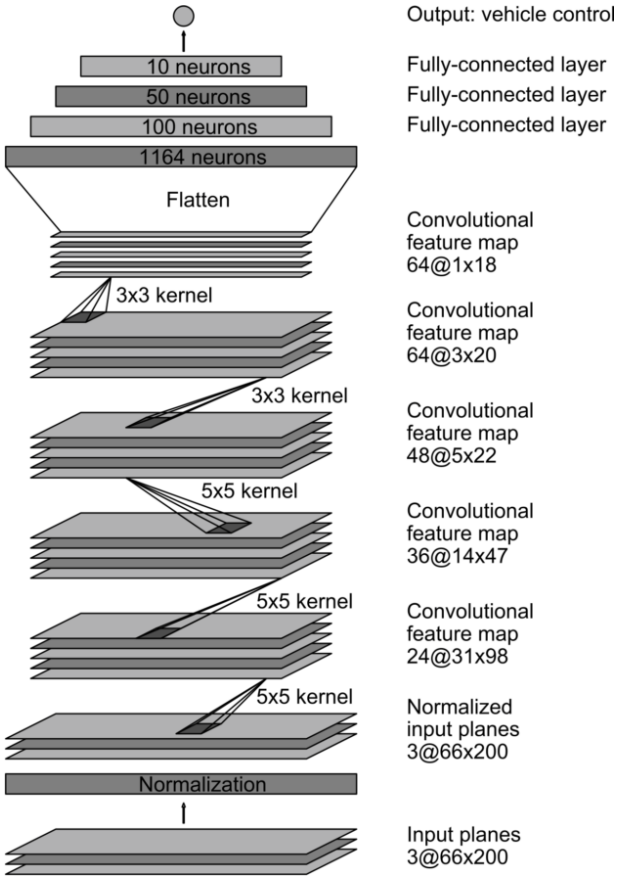

FIG. 1. NVIDIA CNN architecture. The network has about 27 million connections and 250 thousand parameters. [10]

* ppxhl1@nottingham.ac.uk

There are many self-driving car projects that have been done, ranging from large-scale projects like the End-to-End Deep Learning for Self-Driving Cars from Nvidia [10] and the open-source self-driving car project from Udacity [11], to small scale projects like the DeepPiCar project by David Tian [12].

Nvidia's End-to-End Deep Learning for Self-Driving Cars project involves deploying the deep convolutional neural network (CNN) onto a simulator and a real-world car, the results show that CNNs have the capability to learn the task of lane and road following with a small amount of training data [10].

The DeepPiCar project [12] is a scaled-down version of Nvidia's project, instead of deploying the network to a driveable car, it performs self-driving on the SunFounder PiCar-V [1], which is the same car used in this study. The CNN architecture used in both projects is shown in fig. 1

Besides using the Nvidia CNN architecture which is already shown that it can perform self-driving with great accuracy, this project also experiments with other types of CNN architecture to see if the performance can be improved or not.

III. METHODOLOGY

After inspecting the data, there are 5 corrupted image files, the rows corresponding to these files are removed before combining the steering angles and speeds to form a complete dataset. By doing this, the dataset reduced its size from 13,798 to 13,793. Both the steering angles and speeds are normalised to values between 0 and 1. This is achieved by:

$$angle_{norm} = \frac{(angle - 50)}{80} \tag{1}$$

$$speed_{norm} = \frac{(speed - 0)}{35} \tag{2}$$

Researches [13, 14] has shown that performing augmentation on image data can help to reduce overfitting and improve model performance. Image data augmentation involves randomly performing transformations like zooming, panning and blurring on the training data. By applying various transformations to the original images, image data augmentation effectively increases the size of the training dataset. This has the effect of making the model more robust to variations and noise in the input data, which can help prevent overfitting. In addition, the model learns to recognize the underlying patterns in the data, rather than just memorizing the specific examples.

The distribution of the angles in the training data is shown in fig. 2, we can see that the majority of the angles are greater than 0.5 (turning to the right), which means the dataset is unbalanced and can lead to biased or inaccurate model performance and predictions. The effect of this problem can be reduced by randomly flipping the training images horizontally, this simply changes the image from turning right to turning left, and the new steering angle can be calculated by $1 - angle$ while the speed is unchanged. This adds more variations of the training data, specifically the examples of turning left, which is lacking in the original dataset.

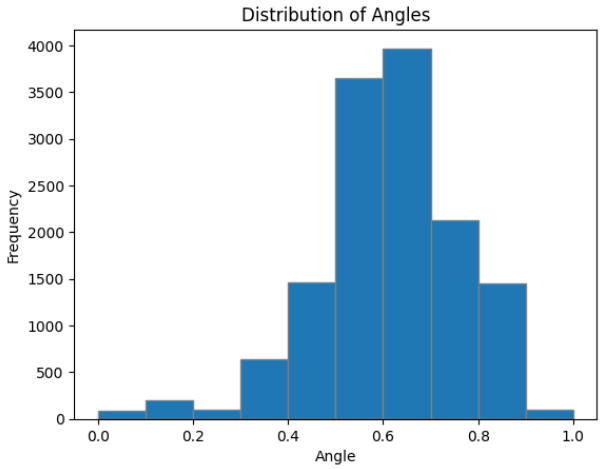

FIG. 2. Shows the angles distribution of the dataset

All the training data are processed by a custom image data generator before being used to train all the models in this project. The custom image data generator includes five operations: zooming, panning, brightness adjustment, blurring and flipping. Each of the operations is applied randomly with a probability of 50%.

A. Models

A total of 6 different Convolutional Neural Networks models were developed and tested, which include 3 custom models that are trained from the ground up and 3 models that utilize a pre-trained network. The data has been split into 80% training and 20% testing before training all the models. The output layer of all the models consists of two dense layers, the first dense layer uses the ReLU activation function to predict a numerical output, which will be the angle. The second dense layer uses the sigmoid activation function to predict a binary output, which is suitable for the speed because it only has values of 0 and 1. Early stopping regularization is implemented in all of the models. When there is no improvement on the validation loss after 10 epochs, the training process will be stopped.

TABLE I. Models Details

Model	Input size	Parameters	Augmentation
Nvidia	200x66	252,486	YES
Nvidia 2	240x180	2,345,286	YES
Custom CNN	240x180	338,742	YES
VGG16	224x224	15,175,391	YES
MobileNet	224x224	3,988,530	YES
MobileNetV2	224x224	3,216274	YES

1. *Nvidia Model*

The first model we developed is the Nvidia Model, fig. 1 shows the model architecture, it is the same model used by Nvidia for the End-to-End Deep Learning for Self-Driving Cars project [10] except we have added a dropout layer after the flattening layer. The model contains 11 layers and has around 27 million connections and 250 thousand parameters. The input image size of this model is 200x66, which is the default input size for the Nvidia model.

2. *Nvidia 2 Model*

The Nvidia 2 model is an alternate version of the original Nvidia model. In this version, we want to experiment with different input image sizes with the Nvidia model, therefore, we have changed the input image size from a scratched 200x66 to 240x180. As a result, the number of parameters for the Nvidia 2 model has increased to around 2 million.

3. *Custom CNN*

The custom CNN is a customized version of the Nvidia 2 model to try to achieve better performance. In the Nvidia and Nvidia 2 models, the stride of the first four 2D convolution layers is set to 2 pixels, while the last two 2D convolution layers is using the default stride value, which will be 1 pixel. The custom CNN changed the stride value for the last two 2D convolution layers to 2 pixels, also the number of neurons for the three dense layers has been changed from 100, 50, and 10 to 128, 64, and 32 respectively. The image input size for this model is kept at 240x180.

4. *VGG16*

The VGG16 model is a custom model built by combining the VGG16 architecture with additional layers. VGG16 is a deep convolutional neural network model for large-scale image recognition, proposed by K. Simonyan and A. Zisserman from the University of Oxford [15]. It has 16 weight layers (13 convolutional and 3 fully connected) and uses 3x3 filters and max-pooling layers with a stride of 2 pixels.

In this model, we replaced the top three 2D convolution layers from the Nvidia model with the VGG16 architecture and load the pre-trained weight into it. An extra dense layer of size 5 was also added before the output layer. Due to the VGG16 being a large model, the number of parameters for this model is over 15 million.

In a typical transfer-learning workflow, training will start by freezing the base model with the pre-trained weights and then fine-tuning with a very low learning rate by unfreezing the base model. Therefore, the model will need to be trained twice, which means it will require a lot of training time. Due to limited computational resources, we have opted for a more lightweight workflow [16]. Instead of training the model twice, we use the pre-trained VGG16 model as a feature extractor, this way we do not need to freeze the base model and the model will only need to be trained once. Feature extraction is an alternate approach to transfer learning, although the performance might not be as high as transfer learning, it provides much faster training time.

5. *MobileNet and MobileNet V2*

Instead of using a large model like VGG16, we think that it will be more appropriate to use a smaller, faster pre-trained model on the SunFounder PiCar-V [1], which has very limited computational power. With that in mind, we have chosen the MobileNet[17] and MobileNet V2[18]. Both model's structure is similar to the VGG16 model except we have replaced the VGG16 pre-trained model with MobileNet and MobileNet V2 respectively.

The difference between the original MobileNet and MobileNet V2 is that the inference time for MobileNet V2 is much faster than the original MobileNet, it also has 30% fewer parameters. Including all the custom layers, the total number of parameters for the MobileNet model is around 4 million, while the MobileNet V2 model has around 3.2 million.

B. Testing

All the models will be compared to each other by speed and angle loss to have a better understanding of the performance of the models.

To evaluate the self-driving capabilities of each of the models, we will be testing them on three different tracks and with 6 scenarios. The tracks layout is shown in fig. 3 and the objective for the car is to stay in the lane while performing the scenarios, which include driving straight on the T-junction track, driving on the oval track, driving on the oval track but in the reverse direction, driving on the figure-of-eight track, driving on the figure-of-eight track but in the reverse direction, stop when a pedestrian

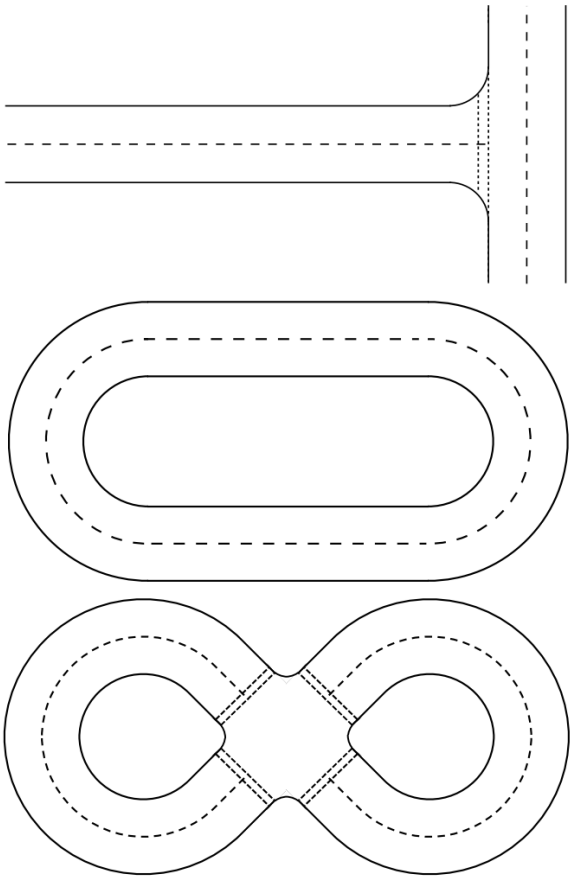

FIG. 3. Showing the T-junction, oval and the figure-of-eight tracks

is in front of the car but keep going when the pedestrian is on the side on the lane.

Research [19] has shown that using the Lite version of TensorFlow instead of the full version of TensorFlow can have a significant performance increase on the MobileNet and MobileNet V2 pre-trained network. Therefore, we have converted the three models that use a pre-trained network to TensorFlow Lite, after that, we compared the inference time of these models with their full TensorFlow and TensorFlow Lite versions.

IV. RESULTS

A. Models Performance

The performance for all the models we developed is shown in table II and fig. 4, different loss functions were used to calculate the angle and speed loss. For angles, it uses the Mean Squared Error (MSE) and for the speeds, it uses the Binary Cross-Entropy. Fig. 4 is a scatter plot that compares both the angle and speed loss for all the models, the closer the model is to the bottom left of the plot, the better it performs. The size of the circle represents the number of the parameters of that model, the larger the circle, the more parameters that the model has.

What we see from table II and fig. 4 is that out of all the models we developed, the model that uses the VGG16 pre-trained network performs the best. We think this can be due to the VGG16 model being our biggest model and having the highest amount of parameters, which enables it to extract more important features from the images.

Our second and third best-performing models will be the original Nvidia model and our Custom CNN. The performance of the two is similar and outperforms both the MobileNet and MobileNet V2 model. Except for the Nvidia 2 model, all of our models performs similarly in terms of the speed loss.

TABLE II. Model Performance

Model	Overall Loss	Angle Loss	Speed Loss
Nvidia	0.0287	0.00920	0.0190
Nvidia 2	0.0360	0.01130	0.2470
Custom CNN	0.0289	0.00940	0.0195
VGG16	0.0171	0.00880	0.0083
MobileNet	0.0230	0.01050	0.0103
MobileNetV2	0.0231	0.01380	0.0093

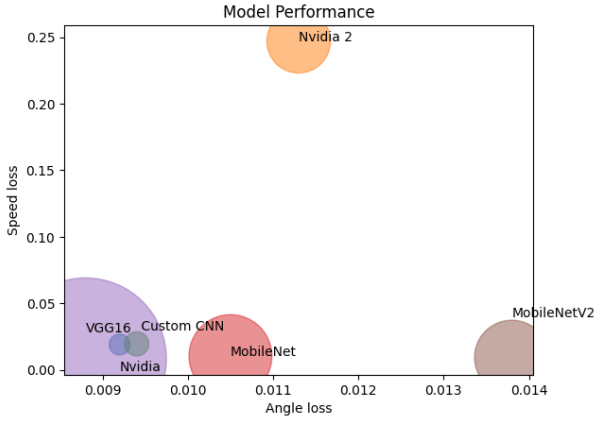

FIG. 4. Showing the performance of all the models by their angle and speed loss.

B. Inference Time

Fig. 5 shows the inference time of the MobileNet, MobileNet V2 and VGG16 model while using TensorFlow and TensorFlow Lite on the PiCar. What we can see is that TensorFlow Lite has a faster inference time across the three models, with the VGG16 model having the longest inference time.

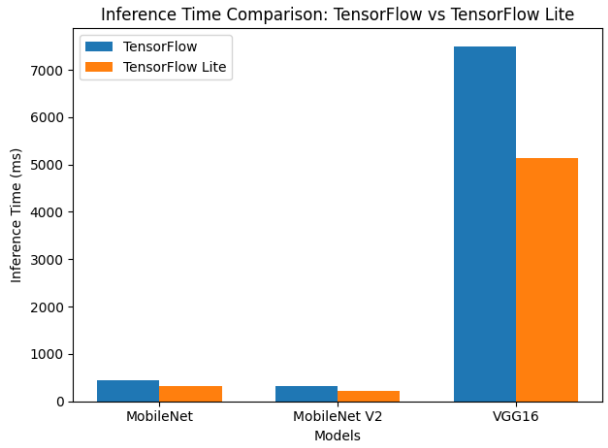

FIG. 5. Shows the comparison of TensorFlow and TensorFlow Lite on three models.

C. Live Testing

The result of living testing is shown in Table III, what we can see is that most of the models are struggling to drive on the reverse oval and reverse figure-of-eight track. The MobileNet V2 model performs the best in live testing, the only scenario that it cannot complete is to drive on a straight line on the T-junction track. We are unsure why the original MobileNet has better performance in terms of angle and speed loss but performs worst than MobileNet V2 in live testing. The VGG16 model has not been used in live testing because the inference time for it is too long and we think that it will be unrealistic to simulate a self-driving scenario with that long of inference time.

V. CONCLUSIONS

In this project, we have developed six deep-learning models that are capable of predicting the steering angle and speed based on an image. In terms of the angle and speed loss, the VGG16 model is performing the best but the inference time is too long that it cannot be used on the Picar. For live testing, the MobileNet V2 model is the best model, it completed 5 out of 6 driving scenarios.

[1] Raspberry pi video car kit (picar-v) for intermediate.

[2] J. M. Anderson, N. Kalra, K. D. Stanley, P. Sorensen, C. Samaras, and T. A. Oluwatola, *Autonomous Vehicle Technology: A Guide for Policymakers* (RAND Corporation, Santa Monica, CA, 2016).

[3] R. Fan, J. Jiao, H. Ye, Y. Yu, I. Pitas, and M. Liu, Key ingredients of self-driving cars (2019), arXiv:1906.02939 [cs.RO].

[4] V. Shreyas, S. N. Bharadwaj, S. Srinidhi, K. U. Ankith, and A. B. Rajendra, Self-driving cars: An overview of various autonomous driving systems, in *Advances in Data and Information Sciences*, edited by M. L. Kolhe, S. Tiwari, M. C. Trivedi, and K. K. Mishra (Springer Singapore, Singapore, 2020) pp. 361–371.

[5] F. Alam, R. Mehmood, and I. Katib, Comparison of decision trees and deep learning for object classification in autonomous driving, in *Smart Infrastructure and Applications: Foundations for Smarter Cities and Societies*, edited by R. Mehmood, S. See, I. Katib, and I. Chlamtac (Springer International Publishing, Cham, 2020) pp. 135–158.

[6] Q. Rao and J. Frtunikj, Deep learning for self-driving cars: Chances and challenges, in *Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems*, SEFAIS '18 (Association for Computing Machinery, New York, NY, USA, 2018) p. 35–38.

[7] D. You, W. Haiyan, and Y. Kaiming, State-of-the-art and trends of autonomous driving technology (2018) pp. 1–8.

[8] V. Shreyas, S. N. Bharadwaj, S. Srinidhi, K. U. Ankith, and A. B. Rajendra, Self-driving cars: An overview of various autonomous driving systems, in *Advances in Data and Information Sciences*, edited by M. L. Kolhe, S. Tiwari, M. C. Trivedi, and K. K. Mishra (Springer Singapore, Singapore, 2020) pp. 361–371.

[9] S. Haj-Assaad, Going "hands free" with gm's super cruise (2021).

[10] M. Bojarski, B. Firner, B. Flepp, L. Jackel, U. Muller, K. Zieba, and D. D. Testa, End-to-end deep learning for self-driving cars (2016).

[11] Udacity, The udacity open source self-driving car project (2021).

[12] D. Tian, Deeppicar-part 1: How to build a deep learning, self driving robotic car on a shoestring budget (2019).

[13] C. Shorten and T. M. Khoshgoftaar, A survey on image data augmentation for deep learning, Journal of big data **6**, 1 (2019).

[14] A. Mikołajczyk and M. Grochowski, Data augmentation for improving deep learning in image classification problem, in *2018 international interdisciplinary PhD workshop (IIPhDW)* (IEEE, 2018) pp. 117–122.

[15] K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition (2015), arXiv:1409.1556 [cs.CV].

[16] fchollet, Keras documentation: Transfer learning amp; fine-tuning (2020).

[17] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications (2017), arXiv:1704.04861 [cs.CV].

[18] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks (2019), arXiv:1801.04381 [cs.CV].

[19] A. Allan, Benchmarking tensorflow and tensorflow lite on the raspberry pi (2023).

TABLE III. Live Testing Results

Model	Straight	Oval	Reverse Oval	Fig. 8	Reverse Fig. 8	Pedestrian
Nvidia	YES	YES	NO	NO	NO	YES
Nvidia 2	NO	NO	NO	NO	NO	NO
Custom	YES	YES	NO	NO	NO	YES
VGG16	n/a	n/a	n/a	n/a	n/a	n/a
MobileNet	NO	YES	NO	YES	NO	YES
MobileNet V2	NO	YES	YES	YES	YES	YES