PHYS4036 DEBUGGATI: CNNS FOR AUTONOMOUS DRIVING Picar

Rajat Goyal[1]

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

Autonomous vehicles are a popular topic in today's age, with every industrial leaders seeking to capitalise on being one of the earliest adopters. Multiple research papers demonstrate that CNNs are the appropriate technology to make this feasible. In this work, multiple CNN architectures were trained and evaluated to determine the model that assists the car in making the optimal decision on its own. InceptionResnetV2 was selected as the final model for the Kaggle competition, whereas MobileNet proved to be the best model for live demo Picar trials due to hardware limitations and shorter inference times.

1. Introduction

Autonomous cars are self-driving vehicles that utilise advanced technologies and AI algorithms to operate without human intervention. The objective is to improve road safety and transportation efficiency. These vehicles rely on sensors, cameras, and AI-based decision-making algorithms to perceive their surroundings and make intelligent choices. Self-driving vehicles have the potential to reduce incidents caused by human error and provide a safe driving experience. To safely navigate roads, they employ computer vision, machine learning, and mapping systems. Existing obstacles include regulations, ethics, cybersecurity, and public acceptability. However, autonomous vehicles have the potential to transform transportation, reduce congestion, and improve accessibility. Continuous development and collaboration are required for their pervasive adoption in a secure manner [1].

2. Background

The remarkable advancements in autonomous car research have been fuelled by the convergence of cutting-edge technologies and the urgent demand for safer and more efficient transportation solutions. Academic institutions, industry leaders, and illustrious research organisations have launched ambitious projects to develop autonomous vehicles that can operate on public roads. Waymo's (Google) published papers on perception, planning, and machine learning algorithms have made significant contributions to autonomous driving research. [2]. With research papers on perception, mapping, and localization, Mobileye, an Intel company specialising in advanced driver-assistance systems, has significantly influenced the development as well [3]. Deep learning for perceptual and simulation-based assessment has been the focus of NVIDIA's research [4]. The Tesla Autopilot system has also made significant advances in autonomous driving technology, although its research papers may not be as widely published. These research efforts have advanced the field of autonomous vehicles, fostering collaboration and the sharing of knowledge while advancing safer and more efficient transportation systems.

3. Methods

3.1. Dataset Information

The dataset was made available on the project's competition website [5]. It had about 14k photos [Figure 1] from all three tracks, including pedestrians, obstructions, traffic and turn signals alongside their target response (speed and steering angle) [5]. The images were 3 channelled RGB of size 240×320. The target labels were normalised to the range 0-1 with speed being either 0 or 1, signifying acceleration or a stop, respectively. Similarly, the steering angle was calibrated within the same range.

$$\text{angle}_{\text{norm}} = \frac{\text{speed} - 0}{35}$$

$$\text{angle}_{\text{norm}} = \frac{\text{angle} - 50}{80}$$

In addition, 1020 test photos were provided to evaluate the trained model's performance. A sample submission csv was also offered to match the submission format. Moreover, almost 500 images were acquired additionally by driving the Picar over the tracks and storing the information.

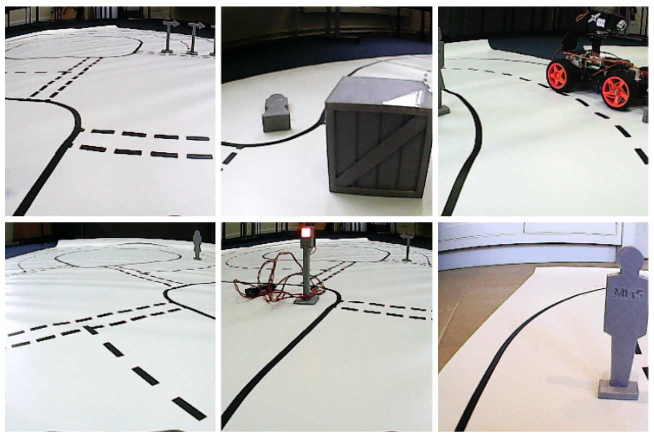

FIG. 1. Sample images in the dataset

3.2. Data Pre-processing

The directories of all training images were stored in one dataframe, while their labels were stored in another. 5 corrupt images were discovered within this dataframe and were consequently removed. But there still remained a discrepancy between directories and their corresponding labels, as the images were being read in a random order. Therefore, a regular expression was used to extract the image names from their respective directories, and the dataframe was sorted accordingly as the image name were their ids. Finally, the corrected directory dataframe was horizontally concatenated with the corresponding labels dataframe.

Then a Tensorflow's flow_from_dataframe generator was created which accepts this dataframe as an input and augments it. While doing so, the input and their corresponding label's column must be specified. Separate train and test generators were created, each of which resized the images to the shape 224×224×3 and rescaled their pixels in the range 0-1 by dividing by 255. In a ratio of 75:25, the train generator was further divided into train and validation.

3.3. CNN

A form of neural network known as a convolutional neural network, or CNN or ConvNet, is especially adept at processing input with a grid-like architecture, such as an image. The layers initially detect lines, curves, and other fundamental patterns, followed by more complex patterns such as faces and objects [6].

3.3.1. Convolutional Layer

The convolution of two functions yields a third function that describes how the shape of one function is altered by another. CNN uses convolution kernels, which are compact 2D matrices convolved over an image. By performing matrix dot product on the input image, a kernel transforms it into a representation that is more comprehensible [6].

3.3.2. Pooling Layer

Pooling is a transformation from vector to scalar, similar to how convolutions operate on each local region of an image. However, it is divided into two categories: max pooling and average pooling. In max pooling, the pixel with the highest intensity is selected and the rest are ignored, whereas in average pooling, the vector is replaced with the average of the pixels [7].

3.3.3. Activation Layer

Activation functions are employed in neural networks to compute the weighted sum of inputs and biases, which is then used to determine whether or not a neuron may be activated [8]. All layers except the final layer use non-linear activation functions because, without them, the model would lack depth and become linear. Depending on the nature of the assignment, the final layer could have linear or non-linear activation functions: linear for regression and non-linear for classification [8].

3.3.4. Batch Normalization Layer

Occasionally, CNNs encounter an internal covariate shift problem during training. It is caused by the variations in data distribution that occur as information traverses the network. The batch normalisation layer is employed to return the batch's mean to zero and standard deviation to one [9].

3.3.5. Global Average Pooling Layer

This layer "flattens" the output of the previous levels into a singular vector that can be used as an input in the fully connected layers that follow. Each value of this vector represents the average value of a specific feature in input map hence resulting in a vector of same size as the depth of input [10].

3.3.6. Fully Connected Layer

A neural network with totally connected layers is one in which each neuron applies a linear transformation to the input vector using a weights matrix [11]. Consequently, all layer-to-layer relationships exist. These layers, which are typically located at the end of the network, learn weights during training in order to yield final predictions [10].

3.3.7. Dropout Layers

Dropout layers terminate random activations by setting them to zero, thus shutting off the respective neurons. This improves the model's ability to generalise and combats overfitting [7].

3.3.8. Loss Functions

The penalty for an inaccurate prediction is known as a loss [10]. It is a score returned by a model that indicates how well or poorly it performed so that adjustments can be made accordingly.

3.4. Transfer Learning

Transfer learning is a technique for machine learning in which the knowledge acquired from training a model on one task is used to improve its performance on a different but related task. Transfer learning allows us to utilise previously trained models that have been trained on large-scale datasets for a different, typically more general task [12].

Transfer learning is based on the idea that models trained on large and diverse datasets acquire complex feature representations that can be applied to a variety of visual tasks even if the target task is distinct from the training task [13].

3.5. Evaluation Metrics

Mean squared error (MSE) was used as metric for Kaggle competition. Group rankings were inversely proportional to their MSE scores. MSE was used as a metric due to its compatibility in regression tasks. It is differentiable, intuitive and penalises larger error more by squaring it [14].

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

For live demo, there were 12 scenarios for test run. The runs included a straight drive and oval drive which also had roadside obstruction or a pedestrian on road as different scenarios. The oval drive was tested in both directions. For the figure of 8 track, the car was tested for a no-stop run, obstruction at junction and whether it stops and resumes at traffic lights. It was also tested for left and right turns on a T-junction and finally ran on max speed of 50 on the oval drive to check the inference time.

Each run was awarded a score out of 3. A score of 3 was given to perfect run, a score of 2 was given to a run with minor inconvenience, a score of 1 was given to a run where car needed to be adjusted manually and 0 was given if the task failed completely.

4. CNN Architectures

Several architectures were researched and compared. In the custom-made architectures, there were minor modifications to the number of convolutional layers and kernel's depth.

4.1. For Kaggle Competition

In the Kaggle competition, six CNN architectures were submitted, with the first four being self-made and the remaining two being transfer learning models.

4.1.1. InceptionResnetV2₁

It is an extension of the Inception architecture which utilises skip connections from the ResNet obtain higher accuracies and better results.

The idea behind the Inception network is to use several "Inception modules" [Figure 2] in its design. On the data it receives, an Inception module does multiple parallel convolutions of different sizes (1x1, 3x3, 5x5) and then concatenates the results together. This lets the network capture both local and global features at different levels of detail, giving a deeper understanding of what was given as input without significantly increasing the number of parameters [15].

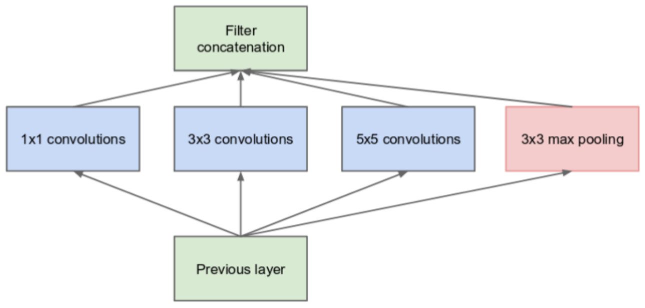

FIG. 2. Inception Modules [15]

This architecture was plagued with vanishing gradients problem hence the use of residual connections [Figure 3] was made. These connections enable the gradient to bypass intermediate network layers and flow directly from one layer to a later layer. This is accomplished by connecting a layer's original input to its output to produce a residual connection. Residual connections facilitate the training of deep neural networks by enabling the network to propagate gradients more effectively, thereby facilitating the successful training of networks with hundreds or thousands of layers [16].

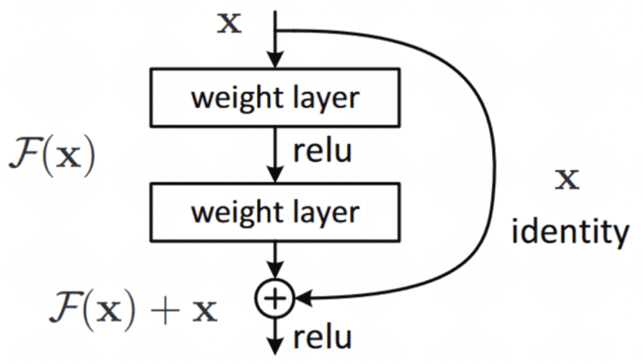
x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

FIG. 3. Skip Connections [16]

Consequently, the combination of these two architectures enables InceptionResnetV2 [Figure 4] to execute convolution operations on multiple parallel paths while avoiding vanishing gradient through the use of multiple inception-resnet modules [17]. The selection of this architecture was motivated by this performance boost without a corresponding increase in computational expense.

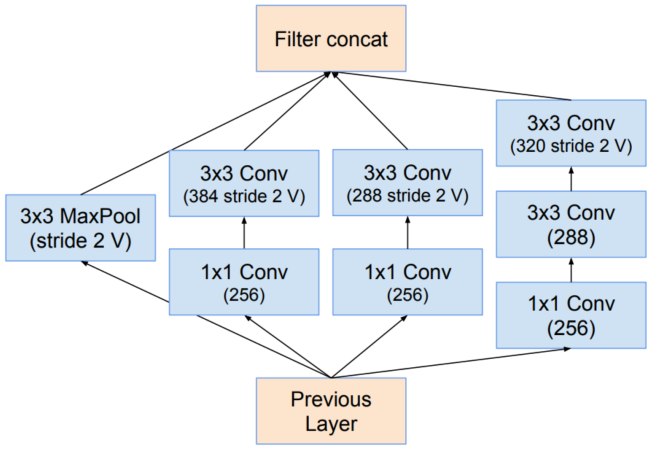

FIG. 4. InceptionResnetV2 Blocks [17]

4.1.2. InceptionResnetV2$_2$

Typically, transfer learning models are used so that pre-trained weights can be imported and applied to the task in order to substantially reduce training time. However, through trial and error and hyperparameter tuning, it was discovered that retraining the layer beyond 745 improved the model's performance, and this new model became the second iteration of InceptionResnetV2 utilized for this project.

4.2. SunFounder Picar

An architecture called MobileNet created by Google in 2017 was deployed for this task. In the machine learning industry, it was a misconception that performance and computational cost are inversely proportional. However, the development of this architecture disproved this prevalent myth. Therefore, this model could operate on low-power mobile devices without sacrificing performance or precision [18].

This model could accomplish this by utilising depth-separable convolutions. The regular convolutional layer is replaced by the application of depth-wise convolution layer followed by point-wise convolution layer. Depth-wise convolution layer uses only 1 kernel. Each channel of the input is convolved over the respective kernel channel and instead of dot product of the results, they are concatenated resulting in a feature map of depth 3 [18]. This output feature map is convolved with a point-wise convolution which has a kernel of shape 1*1*Iz where Iz is the depth of input hence combining the outputs of depth-wise convolution [Figure 5] [19]. MobileNet thus significantly reduces the number of parameters and computations required compared to conventional convolutional layers by segregating the spatial and channel-wise convolutions [18].

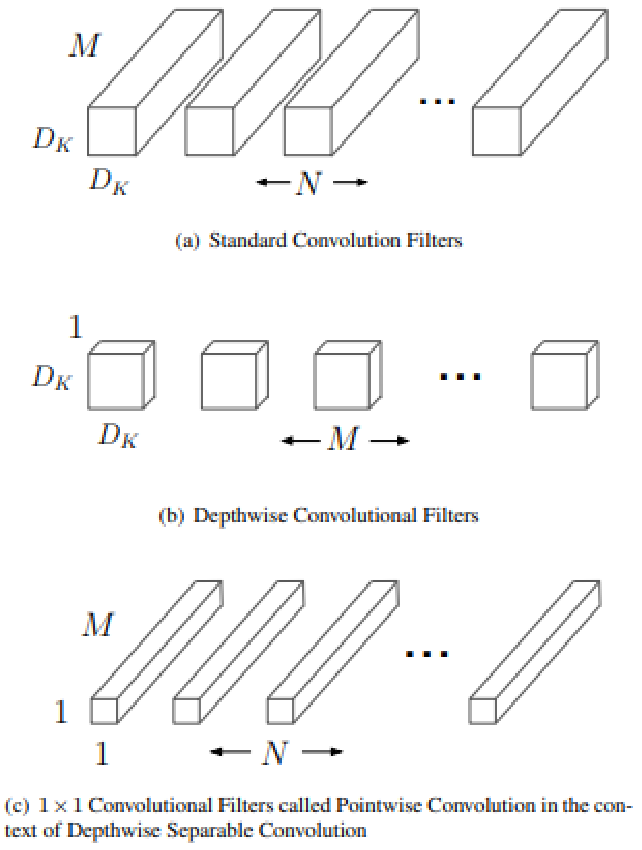
M
D_K
D_K
N
(a) Standard Convolution Filters
1
D_K
D_K
M
(b) Depthwise Convolutional Filters
M
1
1
N
(c) 1 × 1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

FIG. 5. Different types of Convolution [18]

5. Results

A multitude of CNN architectures were trained and tested for both the tasks, the Kaggle competition and SunFounder Picar live demo. After comparing evaluation metrics, best-performing models were selected.

5.1. Kaggle Competition

Models	Public MSE	Private MSE
CNN	2.60071	2.39631
DCNNV1	0.2897	0.30231
DCNNV2	0.27597	0.28387
DCNNV2.1	0.27638	0.28476
InceptionResnetV2$_1$	0.03157	0.02725
InceptionResnetV2$_2$	0.01597	0.01455

TABLE I. Public and Private MSE of CNN models

InceptionResnetV2$_2$ was chosen as the final model for the Kaggle competition because it provided the best evaluation metrics and performed well on both validation and public/private test sets, demonstrating that the model is effective even on unobserved data. With the aid of this model, fourth place in the competition was attained.

5.2. Picar Live Demo

MobileNet was the final chosen model for PiCar's live run.

Track Event	Score
Straight Drive	3
Straight Drive; object at side	2
Straight Drive; pedestrian on track	3
Oval Drive; both directions	1
Oval Drive; object at side	3
Oval Drive; pedestrian on track	3
Figure of 8 drive; no stop	0
Figure of 8 drive; object in junction	3
Figure of 8 drive; traffic light	0
T-junction drive; left turn	0
T-junction drive; right turn	3
Oval Drive; max speed (50)	0
	Total = 21/36

TABLE II. Scores achieved on Picar Live Demo

The car proceeded forward effectively, maintained its lane position, and stopped reliably for all obstacles. On the inner oval track and a portion of the figure-eight track, it failed to maintain its lane. It also failed to turn left, but was able to turn right. The car stopped for the red light but failed to proceed when the green light turned on. The model's inference time was 1200 milliseconds, but it still obtained an overall success rate of 58%.

6. Discussion

Since CNNs extract information from a higher spatial dimension, their already lengthy training times are only made longer by the large dataset (about 14k pictures). Therefore, the decision to use Google Colab platform was made as it provides GPU which helps in reducing the training time. However, the Pro version's 100 compute units were used up in a single day, making the platform of little utility. Kaggle code editor was utilised to resolve this problem because it had enough GPU units.

All images were initially read with opencv imread and stored in a numpy array. However, it decompresses the images as opposed to png's lossless compression. It resulted in a 14GB numpy array that took a lengthy time to save and load, making the model extremely slow. Consequently, research led to the finding of flow_from_dataframe, which is significantly faster and highly compatible with keras, resulting in an improved model even in its later phases.

When the models trained on Kaggle were imported and loaded on the Picar, a bad marshal data (unknown type error) exception was thrown, which prevented any models from being loaded. Further investigation revealed that this error is caused by a version mismatch. The Kaggle platform was running Python 3.7 by default, but was compelled to run Python 3.10, which upgraded the tensorflow and keras packages, thereby resolving the issue.

Immediately following this, a missing file signature error was encountered. The error was caused by inconsistency in the .pb model files that were saved. Somehow, some files were missing from the model when it was downloaded from Kaggle; consequently, the model was retrained, and each file was manually downloaded to resolve this issue.

After resolving all these errors and problems, the deployment of CNNs for training an autonomous Picar for the Kaggle competition and live test run was possible. Due to time constraints and unknown defects on Picar, it was difficult to test different models and incorporate newly collected data. It would have been possible to prevent the high inference time and poor performance of the car in specific cases by training the model on those sections of the track using more images. A lighter model, such as the Nvidia model, was also trained but could not be tested on the car for the reasons stated previously. This limited the car's speed to 30.

7. Conclusion

InceptionResnetV2$_2$ was the final model for the Kaggle competition, while MobileNet was chosen for the Picar live demonstration. A private MSE score of 0.01450 earned fourth place in the Kaggle contest. The live demonstration of Picar was awarded 21 out of 36 points for a success rate of 58%, which was above the class average.

Deep learning is a large degree of trial and error with some applied intuition. Various self-made and pre-trained architectures were applied and tested with extensive hyperparameter tuning. With further research better-fitting model could have been found for both the tasks. In addition, it is important to begin with simpler architectures and ensure that they function before attempting something more complex, as this could result in time and effort loss and further complication of the task. If the issues with Kaggle notebooks had been resolved sooner, more time could have been devoted to improving the performance of Picar, which has vast room for development. In addition, object detection algorithms such as Yolo and image segmentation algorithms such as Mask R-CNN could have been used to further enhance the model and make self-driving more robust by precisely identifying various cases. In light of their success throughout the duration of this undertaking, additional research could also be conducted on various transfer learning model variants.

8. References

[1] S. Singh and B. S. Saini, IOP Conference Series: Materials Science and Engineering **1022**, 012028 (2021).

[2] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov, "Scalability in perception for autonomous driving: Waymo open dataset," (2020).

[3] Mobileye, "True redundancy," (2023).

[4] M. Bojarski, D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," (2016).

[5] A. Moss and M. Lieu, "Machine learning in science ii 2023," (2023).

[6] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, PROC. OF THE IEEE (1998).

[7] J. Olafenwa, "Components of convolutional neural networks," (2018).

[8] L. Panneerselvam, "Activation functions — what are activation functions," (2021).

[9] S. Ioffe, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," (2015).

[10] R. Goyal, "Comp4106 computer vision report: Image classification and segmentation using cnns on oxford flower dataset," (2023).

[11] D. Unzueta, "Fully connected layer vs convolutional layer: Explained — built in," (2022).

[12] S. Bozinovski, Informatica **44** (2020), 10.31449/inf.v44i3.2828.

[13] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," (2014).

[14] R. Tibshirani, Journal of the Royal Statistical Society: Series B (Methodological) **58**, 267 (1996).

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, and V. Vanhoucke, 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) , 1 (2015).

[16] K. He, X. Zhang, S. Ren, and J. Sun, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) , 770 (2016).

[17] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," (2016).

[18] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, and M. Andreetto, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," (2017).

[19] M. Lin, Q. Chen, and S. Yan, "Network in network," (2014).