Autonomous Pi Car: Integration of Deep Learning Models for Self-Driving

Natchira Chongsawad and Jiahui Li

In recent years, autonomous driving has attracted significant interest in both academia and industry, primarily due to advances in deep learning that have fostered innovation in the automotive sector. This research applied supervised deep learning methods on the SunFounder PiCar-V Kit V2 to illustrate complete autonomous driving. In particular, we created a CNN-based multi-output model that allows for complete control by at once predicting angles and speeds. We used a ResNet152V2 backbone for Kaggle competition tasks in order to take advantage of its high-capacity extraction features. For real-world testing, a Transformer architecture based on MobileNetV2 was used, providing a balance between computational efficiency and prediction accuracy. Our method demonstrated the great potential of deep learning solutions in autonomous vehicle development by producing reliable results in both scenario competition and actual driving tests.

Introduction.– Initial designs of autonomous vehicles first appeared in the late 20th century [1], although the idea has been actively researched and improved for many years. However, with to developments in computer processing, machine learning, and sensor technology, notable advances started to be achieved in the early 2000s [1]. One significant event was the DARPA Grand Challenge [2, 3], which drove researchers to create systems that could navigate challenging situations and significantly accelerated the development of autonomous driving technologies. Following that, deep learning became a crucial technique that allowed cars to utilise sensor and visual data to detect and make decisions accurately [4, 5].

Convolutional neural networks (CNNs) and other deep learning models have gained popularity recently as a way to assist systems discover intricate patterns from huge data sets [6]. With the use of these models, autonomous systems can identify objects, analyse road conditions, and make control decisions. Furthermore, this accessibility is essential for training the upcoming generation of researchers.

This project focuses on combining deep learning models into a functional self-driving automobile system using the SunFounder PiCar-V Kit V2 as a platform for investigation and field testing [7]. This method not only demonstrates the importance of deep learning in the development of autonomous car technology, but also supports the broader goal of creating systems that are flexible enough to adapt to the demands of real-world driving situations [8].

Dataset.– This study uses a dataset from the “Machine Learning in Science II 2025” competition, which includes 13,798 images collected in a simulated PiCar environment. Each image is annotated with values representing the car’s angle and speed. the angles indicate at specific values ranging from 50 to 130 degrees in increments of 5, where values less than 90 indicate a left turn, values greater than 90 indicate a right turn, and a value of exactly 90 corresponds to driving straight. The speed is limited to two possible values: 0 or 35 units per second. To prepare the data for training, both the angle and speed values are scaled to fall within a 0 to 1 range using the formulas defined in (1).

$$\text{angle}_{\text{norm}} = \frac{\text{angle} - 50}{80}, \quad \text{speed}_{\text{norm}} = \frac{\text{speed}}{35} \tag{1}$$

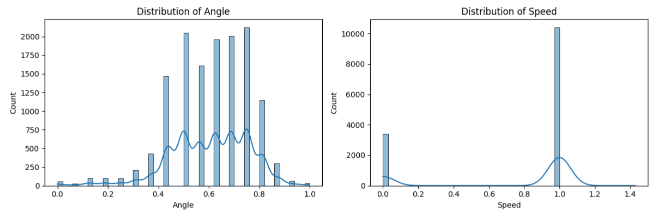

FIG. 1. **Normalized distributions of steering angle and speed in the training dataset.** The left graph illustrates the distribution of steering angles, while the right graph presents the distribution of vehicle speed.

Figure 1 presents the normalised distributions of angle and speed to provide a clearer understanding of the training dataset. These graphs reveal important patterns in driving behavior captured during data collection and highlight the issue of imbalanced data.

The left graph shows that the distribution of angles is skewed towards higher values, indicating a tendency for right-hand turns rather than straight driving or left turns. This follows the convention in which angles above 90 degrees represent right turns. The graph on the right shows a clear bimodal pattern in speed, with most values concentrated at 0 units/sec and 35 units/sec. It also demonstrates that the speed data is discrete, meaning that it represents the vehicle as either moving at full speed or being completely stationary. Furthermore, the data suggests that the vehicle was in motion more frequently than it was at rest.

Related Work.– To enhance the performance of the autonomous driving system in various scenarios, this work adopts concepts and techniques from several existing models. This section presents the methodology used, including data preprocessing, model training, and system evaluation.

ResNet152V2 [9]: The Residual Block structure of a Convolutional Neural Network (CNN) architecture enables the network to train efficiently at a depth of 152 layers while also addressing the vanishing gradient issue. On massive datasets such as ImageNet [10], its improved architecture improves classification accuracy and training stability. ResNet152V2 is used as the foundational model in this study to extract deep features while maintaining picture details. This is an essential foundation for developing a model of autonomous vehicle control that can function in challenging driving conditions.

MobileNetV2 [10]: A Convolutional Neural Network (CNN) architecture that is famous for its excellent efficiency and small size. By using separable convolutions in depth and inverted residual blocks, which significantly reduce the number of parameters and computational cost, it seeks to optimise the efficiency of resources. Because of this, it works well for applications that need to respond in real time. On large-scale image classification tasks, MobileNetV2 maintains high precision despite its light weight. Because MobileNetV2 is suitable with the time and resource restrictions of real-world deployment situations, it was chosen as the basic model for the Live Challenge in this project.

Transformer [11]: A model architecture designed for sequential data processing. The model makes use of a self-attention mechanism that improves the model's effective learning of the relationships between various sequence positions. The Transformer's ability to handle the entire sequence immediately speeds up training and prediction, which is one of its key advantages. Additionally, it is excellent for interpreting relationships in the data in both short and large areas. As part of our work, we use Multi-Head Attention, which is one of the key parts of the Transformer. We set it to use four attention heads and a key vector size of 16. This allows the model to look at different parts of the input sequence at once and understand them from different angles.

Data augmentation [12]: A method that modifies existing photos slightly in order to enhance the quantity of training data. Rotating, flipping, zooming, and adjusting brightness are a few examples of these modifications. By providing the model with more diverse instances, it increases its adaptability and lowers its risk of overfitting. In order to randomly modify brightness, contrast, saturation, and hue, we used TensorFlow functions for data augmentation in this project. These methods enhance model adaptability in practical settings and contribute to visual varie

Gradual Unfreezing [13]: a transfer learning technique in which a pre-trained model's layers are gradually unlocked throughout training. At first, only the top layers are trained, while the earlier ones stay frozen to keep their original knowledge. As training continues, more layers are slowly made trainable, allowing the model to fine-tune deeper features without losing what it had previously learned. This technique helps keep training stable and is especially useful when the available data is limited.

Methodology.– In order to raise the system's overall performance, this section describes the models used for each task as well as the training techniques and setups.
Model Architecture: In this Kaggle competition, no additional image data was collected, so ResNet152V2 was chosen as the base model due to its balanced complexity and strong ability to extract deep visual features from existing image datasets.

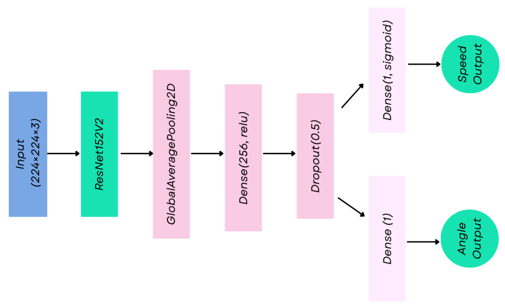

FIG. 2. **Model structure for Kaggle competition.** The model uses a pretrained ResNet152V2 to extract features from input images, followed by shared dense layers and two output branches for angle regression and speed classification.

As illustrated in Figure 2, Starting with a 224 x 224 x 3 input picture, the model architecture uses a pretrained ResNet152V2 network to extract important features. These features are then fed into a GlobalAveragePooling2D layer to reduce the spatial dimensions of the data. The output is passed through a Dense layer with 256 units activated by ReLU, followed by a Dropout layer with a rate of 0.5 to help prevent overfitting. After this shared representation, To manage a multi-task learning configuration, the network splits into two distinct outputs after this shared representation:

- The first branch ends with a Dense(1) layer that predicts the speed as a probability using a sigmoid activation function.
- The second branch concludes with a Dense(1) layer without an activation function to output the angle as a continuous value (regression).

The training process was conducted in two main stages. In the first stage, all layers of the pretrained base model were frozen, and only the custom top layers were trained for 20 epochs. The Adam optimizer was employed with an initial learning rate of 1×10^{-3} to stabilise the learning process during this phase.

In the second stage, fine-tuning was performed by gradually unfreezing the last 50 layers of the base model. To ensure stable weight updates and prevent drastic changes in the learned features, the learning rate was reduced to 0.0005. The second stage of training continued for an additional 10 epochs. Two key callbacks were utilized during training:

ReduceLROnPlateau [14]: This callback dynamically adjusts the learning rate based on the validation loss. If the validation loss does not improve over three consecutive epochs, the learning rate is reduced by a factor of 0.8. After the model reaches a performance plateau, this technique enables more refined updates, improving both stability and convergence.

ModelCheckpoint [15]: This callback was configured to save the model only when it achieved the lowest validation loss. This ensures that the best-performing model on the validation set is retained.

During the live test, an additional 7,500 images were collected. Data augmentation was applied with a 20% probability.

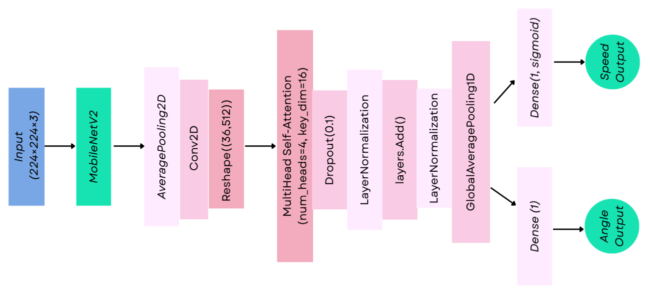

FIG. 3. **Network structure applied in live deployment.** The model uses a pretrained MobileNetV2 for feature extraction, followed by Multi-Head Self-Attention and two output branches for predicting angle and speed.

As shown in Figure 3, the model begins by taking an RGB image input of the same size used in the Kaggle model. This input is passed through a pretrained MobileNetV2, chosen for its lightweight architecture and efficiency in real-time applications on edge devices. The extracted feature maps are then processed using an AveragePooling2D layer to reduce spatial dimensions, followed by a 1×1 Conv2D layer to adjust the channel depth.

The output is reshaped into a sequence of 36 tokens, each with 512 dimensions, and then passed into a Multi-Head Self-Attention module (with 4 attention heads and key dimension of 16). This enables the model to capture complex spatial relationships by attending to different positions in the sequence in parallel.

To improve generalization and ensure training stability, a Dropout layer with a rate of 0.1 is applied, followed by LayerNormalization, a residual connection using Add, and another LayerNormalization layer. GlobalAveragePooling1D is then applied to the sequence to generate a representation with fixed dimensionality.

Finally, the model splits into two output branches, following the same structure as the original Kaggle architecture: one branch for normalized angle regression using a linear activation, and another for stop classification using a sigmoid activation.

The model is trained for 30 epochs using the Adam optimizer. To further enhance training efficiency, a ReduceLROnPlateau callback [14] is used: if the validation loss does not improve for three consecutive epochs, the learning rate is reduced by a factor of 0.8. This adaptive strategy helps reduce the risk of the model getting stuck during training and supports better convergence. To run the model on the PiCar, it was converted from the H5 format to TensorFlow Lite (TFLite) format using TensorFlow Lite version 2.15.0.

Loss function and Metric: This model was applied in both the Kaggle competition and the live challenge. It is designed to produce two parallel outputs: one for predicting angle and speed.

For the Angle Output, which is treated as a regression task, we use the Root Mean Squared Error (RMSE) both as the loss function and the performance metric [16]. RMSE measures how much the model's predicted values differ from the actual values. The formula is:

$$\mathcal{L}_{\text{angle}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(a_i - \hat{a}_i)^2} \tag{2}$$

Here, a_i represents the actual steering angle, $\hat{a}_i$ is the predicted value, and N refers to the total number of samples.

For the speed Output, which is a binary classification task (stop or move), we apply the Binary Crossentropy loss function [16]. We also use Accuracy as a metric to evaluate the model. The loss is calculated using the following equation:

$$\mathcal{L}_{\text{speed}} = -\frac{1}{N}\sum_{i=1}^{N}\left[p_i \log(\hat{p}_i) + (1 - p_i)\log(1 - \hat{p}_i)\right] \tag{3}$$

In this formula, p_i is the actual class label (0 or 1), and $\hat{p}_i$ is the predicted probability, typically produced by a sigmoid function.

The overall loss [17, 18] of the model is calculated by combining the losses from both output branches, as defined below:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{angle}} + \mathcal{L}_{\text{speed}} \tag{4}$$

Results.– After training and optimization, the model achieved strong performance in the Kaggle competition, with consistent results across both the public and private test sets.

TABLE I. The performance of the ResNet152V2 model on the Kaggle leaderboard.

Model	**Private**	**Public**
ResNet152V2	0.01390	0.01043

According to Table I, The results show that the ResNet152V2 model performed well on both the public and private test sets, achieving scores of 0.01043 and 0.01390, respectively. The small difference between these scores suggests that the model generalizes effectively and does not overfit to the public data. Therefore, ResNet152V2 appears to be a reliable and appropriate architecture for this task, providing consistent and accurate results.

As part of the second task, the model was evaluated on the PiCar to test its real-world capabilities. The test included 12 different scenarios aimed at examining how well the model could manage diverse driving situations.

TABLE II. Performance of the MobileNetV2-based model with Self-Attention in real-world tests.

Scenarios	1	2	3	4	5	6	7	8	9	10	11	12	**Total**
Scores	3	3	3	3	3	3	3	3	0	0	3	3	30

The model was evaluated in 12 driving scenarios using the PiCar platform. It achieved full scores (3 points) in 10 scenarios and failed (scoring 0) in 2 specific cases: traffic light scenario and left-turn scenario with directional arrows. Scoring 30 out of 36, the model showed strong practical effectiveness, with only a few failures occurring in more complex or less familiar situations.

Discussions and Conclusions.– According to the experimental results, selecting a suitable model proves to be a major challenge for Kaggle competitions, particularly because no additional data collection is permitted. The core of this work is to identify and train a model that aligns well with the available data.

TABLE III. The performance of various models on the Kaggle leaderboard.

Model	**Private**	**Public**
ResNet152V2	0.01390	0.01043
InceptionV3	0.01622	0.01047
DenseNet201	0.01897	0.01245
Majority voting (3 models)	0.01473	0.01095

The results in Table III demonstrate that ResNet152V2 outperformed the other models on both the private and public leaderboards, even though the selected models were similar in terms of architectural scale and training approach. This observation suggests that ResNet152V2 is not only highly capable but also more effective at generalizing to the dataset, especially within the constraints of data-limited competition settings.

Although large-scale models such as ResNet152V2 are feasible for use in competitions like those on the Kaggle platform, real-world autonomous vehicle applications require more compact and lightweight models. As a result, these smaller architectures may not be capable of interpreting images as deeply as larger ones. When using only MobileNetV2 combined with a custom clustering idea similar to the Kaggle approach, the model was able to complete only straight-road scenarios and the outer path of the oval task. Therefore, balancing the dataset through data augmentation , as illustrated in Figure 5, and incorporating a Transformer module to verify data accuracy are essential steps for improving the model's stability and overall performance.

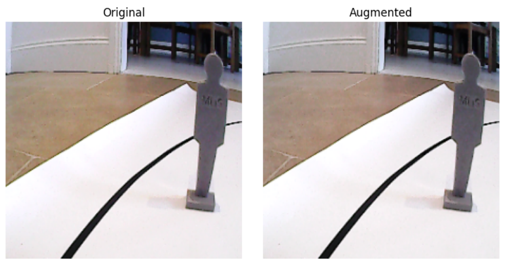

FIG. 4. **Comparison of Original and Augmented Images.** Illustrates the effect of applying brightness, contrast, saturation, and hue transformations. The original image (left) remains untouched, while the augmented image (right) demonstrates visible alterations consistent with the data augmentation process.

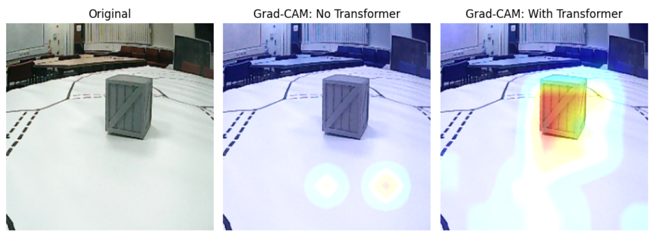

FIG. 5. **Grad-CAM Comparison Between Models With and Without a Transformer.** This figure presents three images that illustrate the model's focus areas: the Original image (left), the Grad-CAM output without a Transformer (middle), and the Grad-CAM output with a Transformer (right). The heatmap highlights the parts of the image that most influence the model's prediction.

Grad-CAM [19] can be used to visualize the areas of an image that the model focuses on during decision-making, helping us understand how well the model interprets a given scene. As shown in Figure 5, The model without a Transformer mainly focuses on the floor, which suggests that it may fail to recognise important objects, such as the crate. In contrast, the image on the right shows that the model enhanced with a Transformer is better at identifying and highlighting the crate. This indicates that incorporating a Transformer improves the model's ability to extract meaningful features and better understand the scene by consistently attending to relevant areas.

In conclusion, this study explored the integration of deep learning models into an autonomous PiCar platform, addressing both competitive and real-world applications. In the Kaggle scenario, a ResNet152V2 model with custom shared dense layers achieved strong performance on both public and private leaderboards. For real-world deployment, a lightweight MobileNetV2-based model enhanced with a Multi-Head Self-Attention (Transformer) block performed reliably, scoring 30 out of 36 across various driving tasks. The findings suggest that large models are suitable for data-limited, accuracy-focused settings, while smaller models are more practical for real-time. A balanced data augmentation strategy—adding roughly 20% variation—proved effective in reducing distribution bias, especially improving performance in turning and obstacle-avoidance tasks. Additionally, the use of attention mechanisms helped the model concentrate on relevant visual regions, leading to better interpretability and decision robustness. These results highlight the importance of aligning model complexity and data strategy to achieve practical and competitive autonomous driving performance.

[1] Ilias Kamal, Khalid Housni, and Youssef Hadi. A survey of autonomous vehicles for traffic analysis. *Indonesian Journal of Electrical Engineering and Computer Science*, 33(2):1016–1029, 2024.

[2] Sebastian Thrun, Michael Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, Melissa Gale, Mike Halpenny, Gabriel Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *Journal of Field Robotics*, 23(9):661–692, 2006.

[3] Chris Urmson, John Anhalt, J Andrew Bagnell, Christopher Baker, Rachel Bittner, Michael Clark, John Dolan, Mark Duggins, Tom Galatali, Christopher Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[4] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[5] Sorin Grigorescu, Bogdan Trasnea, Teodora Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(4):362–386, 2020.

[6] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2722–2730, 2015.

[7] Mark G Bechtel, Cong Liu, and Hyoseung Kim. Deeppicar: A low-cost deep neural network-based autonomous car. In *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 11–21. IEEE, 2018.

[8] Tugrul Sel, Baturay Tatar, Ece Erdem, Cansu Kaleli, and Cagatay Basdogan. Developing a modular package for autonomous driving and experiences gained while implementing this system in a sophomore-level design course. In *Proceedings of the ASEE Annual Conference & Exposition*. American Society for Engineering Education, 2023.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[10] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[12] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[13] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 328–339, 2018.

[14] Rene-Marcel Plonus, Jan Conradt, André Harmer, Silke Janßen, and Jens Floeter. Automatic plankton image classification—can capsules and filters help cope with data set shift? *Limnol. Oceanogr.: Methods*, 19:176–195, 2021.

[15] John Atanbori. SPOTS-10: Animal pattern benchmark dataset for machine learning algorithms. *arXiv preprint arXiv:2410.21044*, 2024. cs.CV.

[16] Juan Terven, Diana-Margarita Cordova-Esparza, Julio-Alejandro Romero-González, Alfonso Ramírez-Pedraza, and E. A. Chávez-Urbiola. A comprehensive survey of loss functions and metrics in deep learning. *Artificial Intelligence Review*, 58:195, 2025.

[17] Han Cao, Sivanesan Rajan, Bianka Hahn, Ersoy Kocak, Daniel Durstewitz, Emanuel Schwarz, and Verena Schneider-Lindner. MTLComb: Multi-task learning combining regression and classification tasks for joint feature selection. *arXiv preprint arXiv:2405.09886*, 2024.

[18] Koushikey Chhapariya, Alexandre Benoit, Krishna Mohan Buddhiraju, and Anil Kumar. A multitask deep learning model for classification and regression of hyperspectral images: Application to the large-scale dataset.

arXiv preprint arXiv:2407.16384, 2024.

[19] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 618–626, 2017.