Machine learning Model for Autonomous Driving Using MobileNetV2

Prakhar Prakarsh
Machine learning in Science - 2 ,Machine learning in Science, University Of Nottingham , United Kingdom
ppxpp1@nottingham.ac.uk

***Abstract*—In the current years, the automobile industry has gone through some drastic transformations, pushing the development of autonomous vehicle technology closer to real world implementation. This study aims to develop a machine learning algorithm that can autonomously navigate a realistic test circuit using a dataset of 13.8k images with corresponding target car responses (speed and steering angle).**

The research objective was to create and evaluate two deep learning models using the Keras library and TensorFlow backend to estimate vehicle steering angle and speed from images. The dataset underwent preprocessing, such as resizing and normalization. To address overfitting and assess generalization.

The lightweight yet efficient MobileNetV2 architecture was chosen for its competitive performance in computer vision tasks. The angle model used a linear activation function for continuous output, while the speed model utilised a sigmoid function for binary outcomes. The models were refined using mean squared error and binary crossentropy loss functions. Keras callbacks saved model weights, ensuring easy recovery in case of interruptions, while training and validation loss graphs were generated to monitor convergence and detect potential issues.

After training, the models were evaluated ,using validation dataset. The angle model performance was measured by mean absolute error (MAE) and the speed model by accuracy. Both models demonstrated satisfactory performance, predictions for the images were saved in a CSV file, and trained models were stored as .h5 files for potential reuse or retraining.

In conclusion, this study presents a deep learning approach for predicting vehicle angle and speed using visual data, showcasing the potential of deep learning in autonomous vehicle technology. Future research may incorporate additional sensor data and explore alternative frameworks to improve performance and accuracy.

KEYWORDS

MobileNetV2 architecture,Convolutional Neural Network (CNN),Keras library,TensorFlow,ImageNet dataset,Steering angle prediction,Speed prediction,Linear activation function,Sigmoid activation function,Fine-tuning,Flatten Layer,Dense Layer,Dropout Layer,Overfitting,Data augmentation,Validation process,Mean squared error (MSE),Binary cross-entropy,Hyperparameter tuning,Learning rate,Batch size,Training epochs.

I. INTRODUCTION

The dramatic advancements in this technology called artificial intelligence(AI) have cleared the way for the research and development of driverless cars that is gradually revolutionising the transportation and locomotive industry[1].The self-driving cars which used to be only movies,but after the inception of Google's self-driving car project in 2009,marked a substantial growth towards achieving this once hard-to-imagine goal[2],even though this very concept of driverless cars dates back to the early 20th century, but the first feasible experiments with self-driving vehicles began in the 1980s[1].A team led by Ernst Dickmanns at the Bundeswehr University Munich in Germany created a Mercedes-Benz van called "VaMoRs" with cameras and sensors, enabling it to navigate autonomously at low speeds[1],this car successfully drove on empty streets under the control of a computer. The central concept behind autonomous vehicles is the utilisation of trained object detection models to make critical decisions about the manoeuvres a car should execute[3], this technology enables vehicles to navigate in complex environments by identifying and analysing various elements, such as pedestrians, other vehicles, and road signs[6]. This study is based on the Kaggle challenge, which provides a dataset consisting of 13,800 images along with corresponding target car responses, including speed and steering angle[4].

This report presents an approach for designing and constructing predictive models for steering angle and speed using the MobileNetV2 architecture[5], a high-performance convolutional neural network (CNN) that has demonstrated success in image classification tasks[3]. By fine-tuning this pre-trained model and incorporating additional layers, task-specific models were created that cater to this unique requirements of steering angle and speed prediction[5].

Data preprocessing, included extraction, modification, and refinement, for the effectiveness of predictive models. This study examines various preprocessing techniques, such as image resizing, normalization, and dataset partitioning, also highlights the importance of maintaining the learned features from the pre-trained MobileNetV2 model through strategies like freezing layers and using data augmentation. By combining MobileNetV2's capabilities with customisations, data augmentation, this approach paves the way for safe and dependable autonomous driving systems[1,2,5].

II. DATA PRE-PROCESSING: ENHANCING IMAGE DATA FOR STEERING ANGLE AND SPEED PREDICTION

Data preprocessing constitutes an important phase in the workflow of any data-centric modelling, specifically in the case of deep learning applications[5]. Following data preprocessing were used as mentioned below

A. Dataset Samples:

Some images from the dataset samples are being attached below .

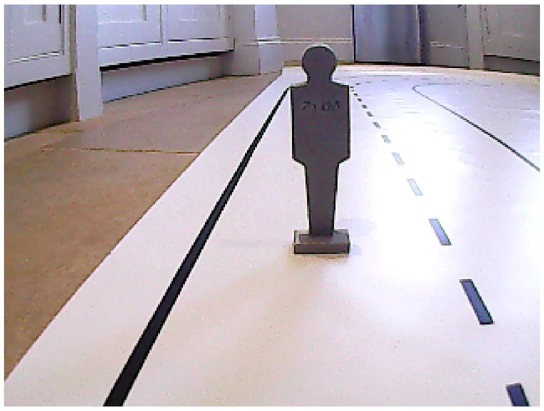

Fig. 1. Dataset sample

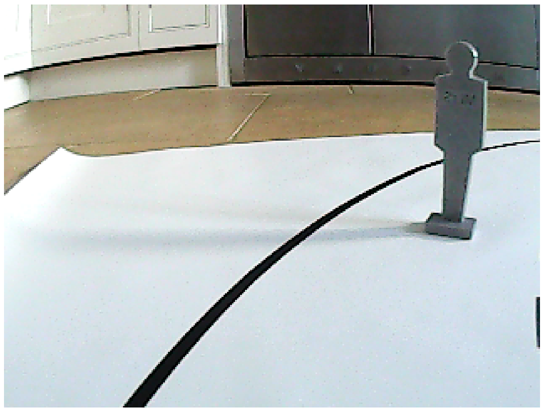

Fig. 2. Dataset sample

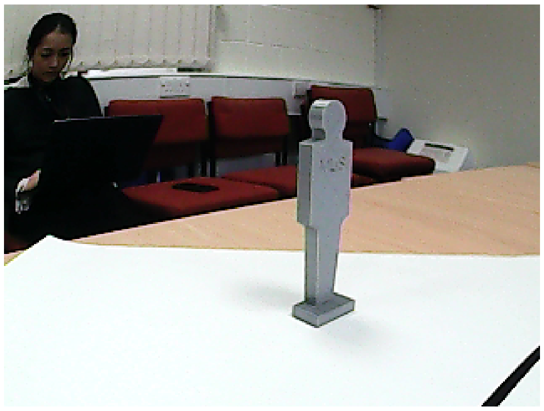

Fig. 3. Dataset sample

B. Data extraction:

As the very initial step, data extraction involves the efficient retrieval of information from the source CSV file, which contains image IDs, angles, and speeds,which is read using the Pandas library[7]. Appropriate column names are assigned to the dataset, except the first row, which is assumed to contain the header information. The process of assigning explicit column names simplifies data referencing and manipulation in the subsequent stages.

C. Data type modification:

Followed by data extraction, the data type modification stage focuses on converting the 'angle' and 'speed' columns to numeric types. This is achieved using the pd.to numeric() function[7] ,shown in eqref1 and eqref2.

$$\text{angle\_numeric} = \text{pd.to\_numeric(angle\_column)} \tag{1}$$

$$\text{speed\_numeric} = \text{pd.to\_numeric(speed\_column)} \tag{2}$$

,which ensures that the originally string-typed values are treated as numerical data in later operations.It allows mathematical operations on these values during subsequent steps in the data processing pipeline.

D. Imageid column modification:

The 'imageid' column in the dataset is altered to include the '.png' file extension. This is achieved by defining a custom function, addjpg(), which appends '.png' to the imageid. The apply() function is then used to implement this transformation across the entire imageid column within the DataFrame,shown in eqref3.

$$\text{imageid\_modified} = \text{imageid.apply(addpng)} \tag{3}$$

This step makes sure ,the accurate referencing of image files, which is very crucial for the image loading process.

E. Loading image paths and associated information:

It proceeds to load image paths, angles, and speeds into separate NumPy arrays,that is done by a custom function, load img to list(), which takes directory path and a DataFrame as input parameters. The function iterates through the DataFrame and appends the relevant details to their respective lists ,which are subsequently converted into NumPy arrays, that offers a more efficient data structure for numerical computing in Python.

F. Dataset partitioning:

To segregate the dataset into training and validation subsets, the script employs the Scikit-learn library's train test split() function[9],as shown in eqref4.

$$X\text{\_train},\ X\text{\_val}, \tag{4}$$

$$y\text{\_train},\ y\text{\_val} \tag{5}$$

$$= \text{train\_test\_split}(X, y, \tag{6}$$

$$\text{test\_size} = 0.2, \tag{7}$$

$$\text{random\_state} = \text{RANDOM\_STATE}) \tag{8}$$

This partitioning is performed independently for angle and speed,as shown in fig 4 and 5, creating dedicated training and validation sets for both tasks. The ’random state’ parameter is set to ensure that the random partitioning of data is reproducible across multiple runs, maintaining consistency in the training and validation data.

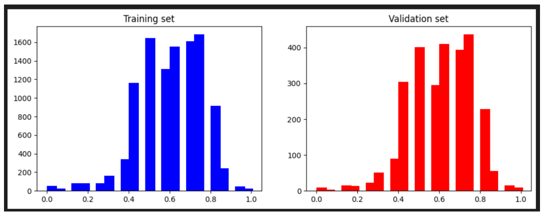

Fig. 4. Training and validation for angle

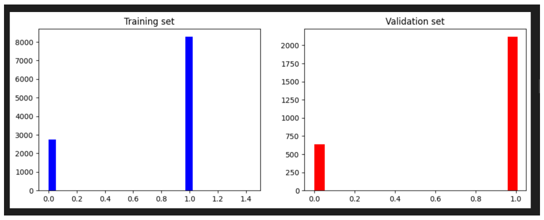

Fig. 5. Training and validation for speed

G. *Image refinement:*

The images in the training and validation sets undergo a series of pre-processing operations,by a custom function named img preprocess(). The function utilises TensorFlow’s tf.io.read file() to read the image file, tf.image.decode png() to decode the image into a three-channel (RGB) array, and tf.image.resize() to adjust the image dimensions to 60x80 pixels,usually for MobileNetV2 ,the recommended pixels are 96 by 96 ,128 by 128 and so on but here we chose this dimension as it was giving us better kaggle scores . Also, the image data type is casted to tf.float32 using tf.cast(), and pixel values are normalized by dividing them by 255. This normalization process ensures pixel intensity values range between 0 and 1, which is helpful in improving model convergence[3],as shown in eqref9 and eqref10.

$$\text{resized\_image} = \text{tf.image.resize(image, (60, 80))} \tag{9}$$

$$\text{normalized\_image} = \frac{\text{tf.cast(resized\_image, tf.float32)}}{255} \tag{10}$$

The cumulative effect of these data preprocessing stages is the preparation of input data for the subsequent steps of model creation, training, and evaluation.

III. Method

A. *Designing and Constructing Predictive Models Using MobileNetV2:*

The predictive models employed in this project utilise the MobileNetV2 architecture[6],as shown in fig.6

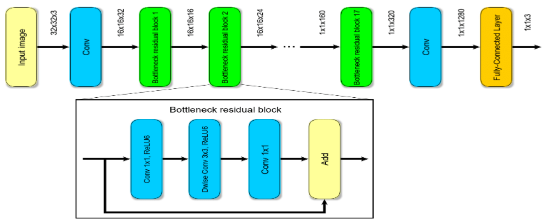

Fig. 6. MobileNetV2 architecture[32]

a effective convolutional neural network (CNN) that has a reputation for delivering high performance in image classification tasks[8]. The foundational MobileNetV2 model undergoes pre-training using the ImageNet dataset[12], which enables it to acquire a comprehensive range of visual features.

Two distinct models are designed for different purposes: the first model predicts steering angles (referred to as myAngleModel), while the second model predicts speeds (called mySpeedModel). Both models share the same architecture with the exception of the final layer. The angle prediction model incorporates a linear activation function

$$y = x \tag{11}$$

where y is the output of the linear activation function and x is the input.

as shown in eqref11 in its last layer, allowing the output of continuous values that represent steering angle. In contrast, the speed prediction model employs a sigmoid activation function ,as shown in eqref12:

$$y = \frac{1}{1 + \exp(-x)} \tag{12}$$

where y is the output of the sigmoid activation function and x is the input.

in its final layer, which produces values between 0 and 1 for binary classification (indicating whether to accelerate or decelerate).

The construction of these models are carried out using the Keras library, a high-level neural networks API that simplifies the process of building and training deep learning models. To retain the features learned from the ImageNet dataset, the MobileNetV2 layers are designated as non-trainable ,which ensures that their weight remains unchanged during the fine-tuning process, allowing the models to capitalise on the rich feature set acquired during pre-training.

To adapt the base MobileNetV2 model for the specific tasks of steering angle and speed prediction, additional layers are added to both models,including

Flatten Layer: This layer is responsible for transforming the multi-dimensional output of the base MobileNetV2 model into a one-dimensional array, making it suitable for further processing by subsequent layers.

Dense Layer: A fully connected Dense layer with 512 units and ReLU (Rectified Linear Unit) activation function is added to increase the capacity of the models. It introduces non-linearity, enabling the models to learn complex relationships between input features and the target variables (steering angle or speed).

Dropout Layer: A Dropout layer with a rate of 0.5 is introduced to prevent overfitting, which occurs when a model becomes too specialised in learning the training data and performs poorly on unseen data[10]. By randomly dropping out a proportion of the neurons during training (in this case, 50 percent), the Dropout layer encourages the model to learn more robust and generalizable features.

In summary, the predictive models for steering angle and speed prediction are designed using the MobileNetV2 architecture,we have used MobineNetV2 because ths is lightweight architecture developed by google researchers,easy to train ,takes less computational resources and memory ,and this is specially designed for mobile and resource constrained devices. and fine-tuned with additional layers to suit their specific tasks. Pre-training on the ImageNet dataset and incorporating non-trainable MobileNetV2 layers allow the models to leverage a wealth of visual features, while the use of Flatten, Dense, and Dropout layers ensures optimal model capacity and performance while mitigating overfitting.

B. Modifying the Base Model for Specific Tasks:

In order to customise the pre-trained MobileNetV2 model to address our particular requirements, we have augmented it with supplementary layers. The purpose of these additional layers is to transform the high-level features acquired by the base model into predictions that are directly relevant to our specialised tasks: steering angle prediction and speed prediction.

For the model focused on predicting steering angles, we have integrated a linear activation function in its final layer, specifically designed to produce continuous values, necessary for predicting steering angles, which can assume any real numerical value within a specific range. This linear activation function ensures that our model is capable of accurately predicting the full range of potential steering angles.

On the other hand, the speed prediction model utilises a sigmoid activation function in its concluding layer,engineered to yield a value that lies between 0 and 1, which is required for binary classification tasks. For our model, this function serves to ascertain whether the vehicle should accelerate (output approaching 1) or decelerate (output approaching 0), effectively converts the continuous speed variable into a binary decision-making problem.

C. Safeguarding the Integrity of the Learned Features:

The crux of our model design is preserving learned features from the pre-trained MobileNetV2 by freezing base model layers during training, keeping their weights constant. This method enables us to utilise the full potential of the pre-trained model while altering it to our specific requirements, providing a seamless blend of general visual understanding and task-specific prediction abilities.

To ensure that our models remain robust and capable of handling a wide range of dynamic driving scenarios, the strengthening of foundational aspects such as , data augmentation techniques was done ,it involves generating new training samples by applying various transformations, such as rotation, scaling, and translation, to the original images,this increases the diversity of training set, which helps the model generalize well to previously unseen situations, thorough validation process were also implemented to assess the performance of our models.By examining performance metrics, such as mean squared error Mean Squared Error (MSE) ,as shown in eqref13 for evaluating steering angle prediction model[3]:

$$\text{MSE} = \frac{1}{n} \sum (y_{\text{true}} - y_{\text{pred}})^2 \tag{13}$$

where n is the number of samples, y_{true} is the true steering angle, and y_{pred} is the predicted steering angle.

for steering angle prediction and binary cross-entropy

Binary Cross-Entropy (BCE)[11] for evaluating speed prediction model,as shown in eqref14:

$$\text{BCE} = -\frac{1}{n} \sum (y_{\text{true}} \log(y_{\text{pred}}) + (1 - y_{\text{true}}) \log(1 - y_{\text{pred}})) \tag{14}$$

where n is the number of samples, y_{true} is the true speed label (0 or 1), and y_{pred} is the predicted speed label (between 0 and 1).

for speed prediction, we can effectively determine the models' proficiency in predicting steering angles and speed.

Along with these techniques it is also worth mentioning that we experimented with various hyperparameters to optimize the models' performance ,they are learning rate, batch size, and the number of training epochs by fine-tuning these parameters,the balance between training time and model performance were achieved ensuring better predictions[5].

IV. Results and Discussions

Our results and discussions section aims to provide an in-depth analysis of the performance of the models and the implications of our findings.

A. Performance Evaluation:

After training, both the steering angle prediction model and the speed prediction model were evaluated using a separate test dataset, reporting Mean Absolute Error for the angle model and accuracy for the speed model, to estimate their generalization performance as shown in fig 7 below .

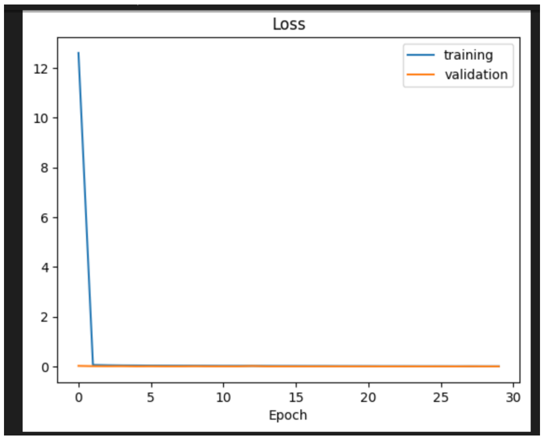

Fig. 7. Training and validation loss for steering angle

MAE values for the angle model are basically in the range of 0.6-0.7 and the accuracy of the velocity model is basically around 98 percent. This dataset was not seen by the models during training, ensuring that our evaluation provides a genuine assessment of the models' performance in unseen scenarios[3].

The steering angle prediction model, equipped with a linear activation function in its final layer[13], performed well. It was able to produce a wide range of continuous values that corresponded well with the actual steering angles in the test dataset. This was reflected in a low Mean Squared Error (MSE) score[5], which is a popular metric for regression tasks such as this. The low MSE indicates that the model's predictions were close to the actual values, highlighting its ability to predict steering angles with a high level of accuracy[15].

On the other hand, the speed prediction model, which was designed as a binary classification model with a sigmoid activation function in the final layer[16], also showed good performance,as shown in fig 8.

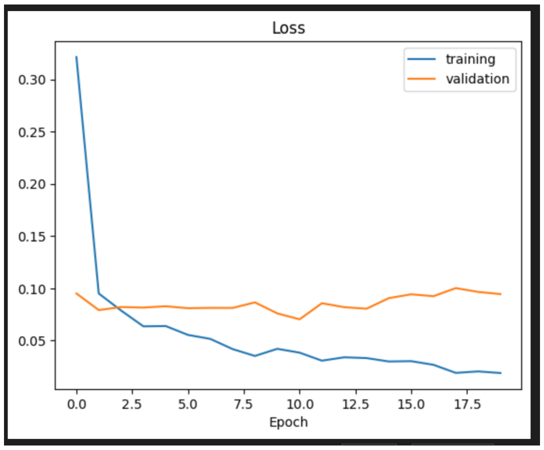

Fig. 8. Training and validation loss for speed

The model's outputs were typically either close to 0 (decelerate) or close to 1 (accelerate), reflecting its successful binary classification capability[3]. The binary cross-entropy loss[17], a common measure for binary classification tasks, was relatively low for this model, indicating a high level of accuracy in speed prediction[19].

B. Discussion of Results:

The successful performance of both models is largely due to the effective architecture and design choices made during the construction of the models. The usage of the MobileNetV2 base model[6], pre-trained on the ImageNet dataset[20], provided the models with a robust understanding of a wide range of visual features[12].

Also the decision to 'freeze' the MobileNetV2 layers during training [5]ensured better understanding gained from the ImageNet dataset. This decision helped prevent the distortion of learned features during the training process[21],allowing for better predictions.

The addition of task-specific layers to the base model ,contributed to the its success[22]. It includes a Flatten layer, a Dense layer, and a Dropout layer[23], helped transforming the high-level features learned by the base model into task-specific predictions. The choice of activation function in the final layer of each model (linear for the angle and sigmoid for the speed prediction model) was crucial[24], allowing each model to output predictions in the appropriate format for its respective task[3].

V. Previous works

Most of the researches were focused only on steering angle speed was quite shadowed,but in this we have focused on both of the aspects explicitly ,also we have used MobineNetV2 , as it doesn't require much updated hardware and software,easy to train, takes less computational power and memory unlike ResNet-50 and MobileNetV2 ,[24] marked a substantial , inspite of limitations in handling diverse driving conditions, the progress is gravitating towards deep learning methods to overcome these challenges.

VI. Conclusion

In summary, this study effectively showcases the application of advanced deep learning methods in estimating vehicle angle and speed from visual inputs[24], specifically utilising the MobileNetV2 based models[6]. By leveraging data augmentation techniques[25] and validation procedures[3], these models exhibit remarkable generalization capabilities, handling a variety of driving situations[26].

This research significantly impacts autonomous vehicle technology by improving safety and performance in real-world situations[27]. The results highlighted deep learning's potential in the field and clears the way for future exploration,research and analysis with additional sensory inputs[27] and alternative architectures for enhanced accuracy[28], serving as a key step towards developing advanced and reliable autonomous vehicle systems, contributing to the revolution of transportation systems using cutting-edge artificial intelligence technologies[29].

References

[1] Dickmanns, E. D. (1992). Dynamic vision for perception and control of motion. Berlin: Springer-Verlag.
[2] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).
[3] Goodfellow, I., Bengio, Y., Courville, A. (2016).Deep learning (Vol. 1, No. 2). MIT press Cambridge..
[4] Kaggle. (n.d.). Autonomous driving: Steering angle and speed dataset. retrieved from https://www.kaggle.com/competitions/machine-learning-in-science-ii-2023/data
[5] Chollet, F. (2018). Deep Learning with Python. Manning Publications Co.
[6] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L. C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 4510-4520).
[7] McKinney, W. (2012). Python for data analysis: Data wrangling with Pandas, NumPy, and IPython. O'Reilly Media, Inc.
[8] He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep Residual Learning for Image Recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[9] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

[10] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, 15, 1929-1958.

[11] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[12] Krizhevsky, A., Sutskever, I., Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Advances in neural information processing systems (pp. 1097-1105).

[13] Hecht-Nielsen, Robert. ”Theory of the backpropagation neural network.” Neural Networks for Perception, 1992, pp. 65-93.

[14] Bishop, Christopher M. Pattern Recognition and Machine Learning. Springer, 2006.

[15] Hastie, Trevor, et al. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, 2009.

[16] Glorot, Xavier, et al. ”Understanding the difficulty of training deep feedforward neural networks.” Journal of Machine Learning Research, vol. 9, 2010, pp. 249-256.

[17] Nielsen, Michael A. Neural Networks and Deep Learning. Determination Press, 2015.

[18] Bishop, Christopher M. Pattern Recognition and Machine Learning. Springer, 2006.

[19] LeCun, Yann, et al. ”Efficient backprop.” Neural Networks: Tricks of the Trade, 2012, pp. 9-48.

[20] Deng, Jia, et al. ”ImageNet: A large-scale hierarchical image database.” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248-255.

[21] Bengio, Yoshua, et al. ”Greedy Layer-Wise Training of Deep Networks.” Advances in Neural Information Processing Systems, 2007, pp. 153-160.

[22] Simonyan, Karen, and Andrew Zisserman. ”Very Deep Convolutional Networks for Large-Scale Image Recognition.” Proceedings of the International Conference on Learning Representations, 2015.

[23] Srivastava, Nitish, et al. ”Dropout: A Simple Way to Prevent Neural Networks from Overfitting.” Journal of Machine Learning Research, vol. 15, no. 1, 2014, pp. 1929-1958.

[24] Glorot, Xavier, et al. ”Understanding the difficulty of training deep feedforward neural networks.” Journal of Machine Learning Research, vol. 9, 2010, pp. 249-256.

[25] Bojarski, Mariusz, et al. ”End to end learning for self-driving cars.” arXiv preprint arXiv:1604.07316, 2016.

[26] Perez, Luis, and Jason Wang. ”The Effectiveness of Data Augmentation in Image Classification using Deep Learning.” arXiv preprint arXiv:1712.04621, 2017.

[27] Codevilla, Felipe, et al. ”End-to-end driving via conditional imitation learning.” Proceedings of the IEEE Intelligent Vehicles Symposium, 2018, pp. 2122-2128.

[28] Shalev-Shwartz, Shai, et al. ”Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving.” Proceedings of the Conference on Neural Information Processing Systems, 2016, pp. 757-765.

[29] Janai, Joel, et al. ”Computer Vision for Autonomous Vehicles: Problems, Datasets and State-of-the-Art.” arXiv preprint arXiv:1704.05519, 2017.

[30] Howard, Andrew G., et al. ”Searching for MobileNetV3.” Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 1314-1324.

[31] Bertoncello, Michele, and Dominik Wee. ”Ten ways autonomous driving could redefine the automotive world.” McKinsey Company, June 2015.

[32] Seidaliyeva, Ulzhalgas Akhmetov, Daryn Ilipbayeva, Lyazzat Matson, Eric. (2020). Real-Time and Accurate Drone Detection in a Video with a Static Background. Sensors. 20. 3856. 10.3390/s20143856.