Using Convolutional Neural Networks for Object Detection in Autonomous Driving

Salaar Mir[1]

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

(Dated: May 17, 2024)

Recent developments in computer vision and machine learning technologies have caused rapid advancements in the field of autonomous driving. The ability of such models to accurately and efficiently detect important features of input data in real-time has caused for the evolution of powerful self-driving systems. This study develops two multi-output Convolutional Neural Networks (CNNs) to predict the angle and speed of input images for real-time and simulated autonomous driving tasks. Transfer learning is used to build upon the MobileNetv2 CNN architecture and further optimise it for the purposes of object detection during autonomous driving. One model was used for a Kaggle competition, where it had to correctly predict angle and speed outputs for a series of test images. The other model was used for a real-time task, requiring the need for efficient predictions. Results indicated the real-time model achieved better generalization with reduced overfitting compared to the Kaggle model, demonstrating the effectiveness of model simplifications for real-time applications.

I. INTRODUCTION

The field of autonomous driving has been rapidly evolving, with significant advancements driven by the integration of machine learning and computer vision technologies. A critical aspect of such vehicles is their ability to detect and recognize objects in real-time, which ensures safe navigation and decision-making. CNNs have been at the forefront of these advancements due to their ability to learn and extract hierarchical features from image data effectively. Previous studies have demonstrated various approaches to enhance object detection for autonomous driving. Li et al. built a confidence-aware object detection model upon the MobileNetv2 [1] architecture using Gaussian distributions to create bounding box labels for more accurate recognition of objects encountered in real-life situations [2]. Further work done by Uçar et al. combined the existing AlexNet [3] CNN architecture with a Support Vector Machine (SVM) for improved feature extraction and more accurate pedestrian recognition [4].

This report explores the use of CNNs, particularly leveraging the MobileNetV2 architecture, for object detection in autonomous driving. It details the methodologies employed, including the use of transfer learning, data preprocessing, and model training. The performance of two models for distinct scenarios is compared to highlight the effectiveness of these approaches in creating robust and efficient object detection systems essential for autonomous driving technology.

II. RELATED WORK

A. CNNs

CNNs have become an essential part of successful autonomous driving applications. These neural networks are feedforward networks designed to process multiple arrays as input (such as RGB image arrays). They consist of three primary layers: convolutional, pooling, and fully connected. Convolutional layers extract local feature combinations, generating feature maps that retain the spatial structure of the input. Pooling layers downsample these feature maps, preserving important information while detecting commonalities among features. Fully connected layers are usually the final layers before the output and are responsible for flattening the input from previous layers, preparing it for the output layer [5]. The MobileNetv2 architecture is specifically useful for real-time tasks due to its increased efficiency, allowing for lower inference times during the task. The model utilises depth-wise and point-wise convolutional blocks to reduce computation time in comparison to traditional convolutional layers [6].

B. Transfer Learning

Transfer learning is the process of utilising the knowledge of a previously trained model to improve or adapt a new model on a related task [7]. This approach leverages the knowledge acquired by the original model during its training on a large dataset, significantly reducing the need for extensive data and computational resources for the new task [8]. In the context of object detection for autonomous driving, transfer learning is particularly beneficial as it allows the utilisation of sophisticated models trained on vast image datasets, such as ImageNet [9].

III. METHODS

A. Model Architecture

Two separate models were constructed for this project, one for a live testing session where the car needed to predict the angle and speed of images in real-time. The other

model was used for an online Kaggle competition where the model was simply given a series of test images and had to correctly predict for speed and angle. We used the transfer learning methodology to build upon the existing MobileNetv2 CNN architecture and fine tune it for the purposes of object detection in autonomous driving. Although the Kaggle competition did not require low inference times and efficient computation, we still found that a fine-tuned MobileNetv2 model was exceptionally accurate for the task..

Though the same base model was used for both competitions, the model architecture was quite different.

1. Kaggle Model

The Kaggle model involved significantly more computation due to the use of larger layers with more units.

After the initial layers of the MobileNetv2 model, a Global Average Pooling (GAP) was used to down-sample the spatial dimensions of the feature maps extracted by the convolutional layers of the base model. This allowed for more compact and efficient representations of the feature maps to be passed onto the next layers.

A 1024-unit dense layer was incorporated to capture complex, high-level features by integrating the previously reduced feature maps. The large number of units allow the network to distinguish between subtle differences in the data, allowing for more accurate object detection.

Next, a dropout layer was also added with a rate of 0.5. This was done to mitigate overfitting by randomly setting 50% of the input units to zero during each forward pass of training. This prevents the model from relying too heavily on specific neurons, promoting the learning of more robust features.

Finally, a batch normalisation layer was also added to stabilise and accelerate the model training. By normalising the inputs to each layer, they maintain consistent distributions which help mitigate against vanishing and exploding gradients [10].

2. Real-time Model for Live Testing

The model used for the live testing had to be significantly simplified to allow for lower inference times. Although the overlying structure is quite similar to the Kaggle model, certain features were modified. The dense layer only consisted of 52-units, to allow for more efficiency and less overfitting. Despite the Kaggle model being able to perform well on the test data for the competition, it seemed to be somewhat overfitting to the training images. For this reason, a smaller dense layer limited the models capacity to memorise the training data.

The dropout rate for this model was also decreased from 0.5 to 0.3. This adjustment allows the smaller model to retain more information during each training iteration, as a dropout rate of 0.5 might have been too aggressive.

Finally, the batch normalisation layer was removed. This was done to lessen the inference latency and reduce the computational load.

Both models used a ReLU activation function to introduce nonlinearity for the dense layer, while softmax and sigmoid activation functions were used for the angle and speed outputs respectively.

Figure 1 shows the underlying architecture and discussed layers of the real-time model.

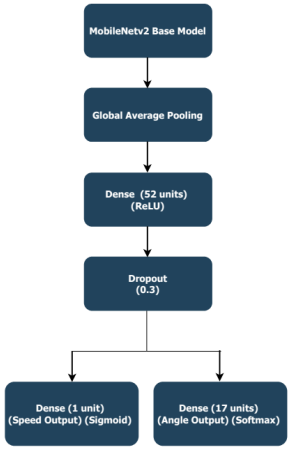


FIG. 1: **Real-time CNN model architecture.** Showing the various layers that were built upon the MobileNetv2 base model.

B. Data Preprocessing

The techniques used for data preprocessing were similar for both models. Input images were first converted from an RGB to YUV matrix, followed by the addition of Gaussian blur, and finally rescaled to 192x192 pixels.

Conversion to a YUV matrix was useful to separate the image brightness (Y channel) from its color information (U and V channel). In lane detection tasks, there is usually high contrast between the road markings and surface. Thus, the luminance information of images is useful to make the detection of lane lines more robust against variations in color and lighting conditions [11]. Figure 2a shows an example image used for training, highlighting the contrast between the lane markings and road surface.

Additionally, Gaussian blur was added to the images to reduce high-frequency noise by averaging across pixels within a neighborhood [12]. This technique is also useful to make the model more robust to motion blur during live testing [13].

Finally, the image was rescaled to a size of 192x192 pixels to reduce computation and inference time. Larger images contain more pixels, increasing the number of computations for each layer in the network. These images also require more memory to store and process, impacting both GPU memory and memory bandwidth, increasing overall computation as well as inference time.

C. Data Augmentation

Despite similar augmentation techniques, the augmentation process for both models was different. For the Kaggle model, offline augmentation was implemented, focusing on the identification and augmentation of images within minority classes before the training procedure started. Conversely, the real-time model used online augmentation, allowing images to be augmented directly during training. Online augmentation was used for this model so that there was no longer a static dataset but rather one that was augmented in real-time. This allowed for virtually infinite types of data variations, allowing for the model to be more robust during the live testing [14]. However, this did mean there were still class imbalances within the data.

The actual augmentation techniques included applying random brightness and contrast perturbations to the images to make the model more robust to noise. Furthermore, random horizontal flipping was also applied to minority classes for the Kaggle model, however not implemented for the real-time model. This was to reduce inference time, as horizontal flipping also requires modifying the labels of the data, adding further computation. Figure 2 shows an example training image before and after data augmentation. The image in Figure 2b looks distinguishably more noisy and unclear.

D. Training Procedure

During training, the total 13798 training images were split into training and validation sets and organised into batches of 128 images to be processed and evaluated. The output for both angle and speed were treated as classification problems, where angle could be classified into 17 different labels, and speed was binary. Angle labels were converted to numerical representations using one-hot encoding. This technique creates binary vectors for each category, ensuring that the machine learning algorithms can process the data without inferring any ordinal relationships among labels [15].

To optimise the model during the training process, categorical cross-entropy and binary cross-entropy were used as loss functions for the angle and speed respectively. Categorical cross-entropy is useful for multi-class classification problems to measure how well a machine learning model's predicted probabilities match the true class labels. The equation for this is given by:

$$\text{Loss} = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic}\log(p_{ic}) \tag{1}$$

Where:

- N is the number of images.
- C is the number of classes.
- y_{ic} is a binary indicator (0 or 1) if class label c is the correct classification for image i.
- p_{ic} is the predicted probability that image i belongs to class c.

This function sums over all classes and images, penalising the model based on the log likelihood of the correct class's predicted probability. Therefore, a higher probability associated to the correct class results in a lower loss.

Binary cross-entropy is similar to this loss function, however it is used for classification problems involving 2 possible classes. The equation for it is given by:

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i\log(p_i) + (1-y_i)\log(1-p_i)\right] \tag{2}$$

Where:

- N is the number of images.
- y_i is the true binary label (0 or 1) for image i.
- p_i is the predicted probability that image i belongs to the positive class.

This function is similar to that of categorical-cross entropy, but adjusted to only account for 2 possible classes.

To optimise the loss function, Root Mean Square Propagation (RMSProp) was chosen for the Kaggle model because of its adaptive learning rate and stable convergence [16]. However, this was changed to Adaptive Moment Estimation (Adam) for the real-time model due to its combination of adaptive learning rate and momentum, allowing for more robustness and efficiency [17].

Additionally, accuracy was used as the evaluation metric for both angle and speed output. This metric measures the proportion of correct predictions out of the total number of predictions. This metric was chosen because it provides the clearest indication of how well the model would perform on unseen images, both for the Kaggle competition as well as during live testing.

E. Training Phases and Finetuning

As mentioned previously, this project utilised transfer learning, meaning that the training procedure involved two main phases: an initial training phase and a fine-tuning phase. Both phases were run over 10 epochs. During the initial training phase, the pre-trained model's weights were kept fixed, and only the newly added layers were trained on the new dataset. This helps adapt the model to the specific features of the new task without disturbing the learned features of the base model. In the subsequent fine-tuning phase, the entire model, including the pre-trained layers, was trained on the new dataset but with a much lower learning rate. This allows

(a) (b)

FIG. 2: **Initial and augmented training image example.** (a) Original training image before augmentation. (b) Augmented image after random contrast and brightness perturbations. Augmented image is more noisy and unclear.

for fine adjustments to the pre-trained weights, ensuring the model can generalise well to the new task. For the initial training phase, the learning rate of the optimiser was set to 0.001, while for the finetuning phase it was set at 0.0001.

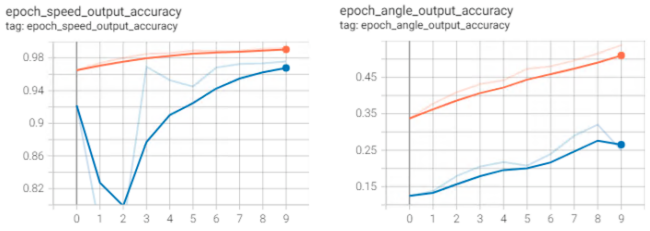
F. Hardware

The main body of the car used to run the real-time CNN model was a SunFounder PiCar-V kit V2 [18] which was equipped with a Raspberry Pi (RPi) 4. The car was preassembled and calibrated, requiring for the model to be loaded onto it to be run.

IV. RESULTS

A. Kaggle Model

The model created for the Kaggle competition ended up coming 9th overall on the day, performing quite well against the test images. The angle accuracy against the validation data was around 32.67%, while the speed accuracy got up to 96.78%. Figures 3a and 3b show the training and validation accuracy scores of the speed and angle outputs of the model, respectively. The blue lines show the scores of the validation data, while the red lines show scores for training. These results were taken from the finetuning phase of training. Despite the impressive accuracy scores, there was some observable overfitting, as the training scores were consistently higher than the validation scores.

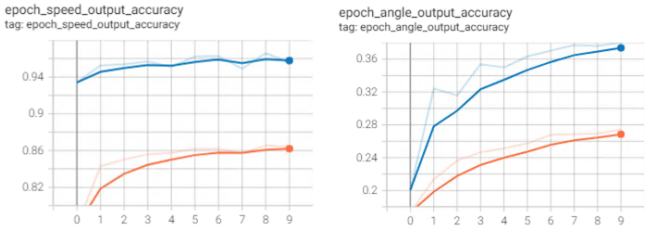


(a) (b)

FIG. 3: **Kaggle model speed and angle accuracy.** (a) Showing training and validation scores of speed accuracy as training epochs increase. (b) Showing training and validation scores of angle accuracy as epochs increase. Blue line shows validation performance while red is training.

B. Real-time Model

The real-time model used for live testing ended up performing very well, ultimately coming 4th in the competition. Figures 4a and 4b show the training and validation accuracy scores of the speed and angle outputs of the model, respectively. The blue lines show the scores of the validation date, while red lines show scores for training. The angle accuracy against the validation data was around 37.38%, while the speed accuracy got up to 95.79%.



(a) (b)

FIG. 4: **Real-time model speed and angle accuracy.** (a) Showing training and validation scores of speed accuracy as training epochs increase. (b) Showing training and validation scores of angle accuracy as epochs increase. Blue line shows validation performance while red is training.

C. Model Comparison

The accuracy scores of the model used for live testing show that there seemed to be less overfitting to the training data, as validation scores were consistently higher than training scores. As mentioned previously, several adjustments were made to the Kaggle model to mitigate against its overfitting. The model architecture was significantly simplified, using smaller dense layers as well as less neurons being dropped out during each training iteration. Furthermore, the learning rate of the optimiser was also reduced during the finetuning phase of training, allowing for more controlled updates of the model weights. These adjustments also caused for lower inference times. Additionally, the real-time model was also converted to a TensorFlow Lite (TFLite) [19] model, to allow for more efficiency and further reduced inference times. These models use quantisation techniques to reduce model size and enhance performance. Such techniques allow for the conversion of 32-bit floating point weights into 8-bit integers, making the models significantly smaller. TFLite models also require less computational power and memory bandwidth, leading to faster inference times [20].

V. CONCLUSION

In this work, we compared the architecture of two distinct CNNs built upon the MobileNetv2 base model. One model was optimised for the purposes of a real-time autonomous driving task, while the other was tested on a series of online input images for a Kaggle competition. The Kaggle model demonstrated overfitting to the training data and had to be simplified and made more efficient to perform well on real-time data during the live task .

To improve upon the model, data augmentation techniques could be focused on minority classes, allowing for more balanced training. An example method that could be used to do this is synthetic minority over-sampling technique (SMOTE) [21], which uses a k-nearest neighbours algorithm to sample data from minority classes and create synthetic examples using interpolation.

To further extend the project, it would be interesting to implement other models such as Vision Transformers (ViTs) or Reinforcement Learning (RL) algorithms to see whether they perform up to the standards of advanced CNNs. Recent work done by Li et al. has explored the use of plain ViTs for the purposes of object detection [22]. It would be fascinating to further finetune such models for autonomous driving tasks. Additional work done by Wang et al. has also looked at the use of efficient RL models for self-driving systems [23]. The code used in this paper is open-source, so potential future steps could include comparing such models to ours to see whether both can perform up to the same standard.

[1] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018).
[2] W. Li and K. Liu, Confidence-aware object detection based on mobilenetv2 for autonomous driving, Sensors (Basel, Switzerland) **21** (2021).
[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, Imagenet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems*, Vol. 25, edited by F. Pereira, C. Burges, L. Bottou, and K. Weinberger (Curran Associates, Inc., 2012).
[4] A. Uçar, Y. Demir, and C. Güzeliş, Object recognition and detection with deep learning for autonomous driving applications, SIMULATION **93**, 759 (2017), https://doi.org/10.1177/0037549717709932.
[5] K. O'Shea and R. Nash, An introduction to convolutional neural networks (2015), arXiv:1511.08458 [cs.NE].
[6] 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, CoRR **abs/1704.04861** (2017), 1704.04861.
[7] S. J. Pan and Q. Yang, A survey on transfer learning, IEEE Transactions on Knowledge and Data Engineering **22**, 1345 (2010).
[8] Y. Wu, T. Li, L. Wang, M. Liu, and Y. Tian, Efficient online transfer learning for 3d object classification in autonomous driving, arXiv preprint arXiv:2104.10037 (2021).
[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in *2009 IEEE Conference on Computer Vision and Pattern Recognition* (Ieee, 2009) pp. 248–255.
[10] S. Ioffe and C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift (2015), arXiv:1502.03167 [cs.LG].
[11] A. D. Storsæter, K. Pitera, and E. McCormack, Camera-

based lane detection—can yellow road markings facilitate automated driving in snow?, Vehicles **3**, 661 (2021).

[12] J. M. Blackledge, Chapter 16 - segmentation and edge detection, in *Digital Image Processing*, Woodhead Publishing Series in Electronic and Optical Materials, edited by J. M. Blackledge (Woodhead Publishing, 2005) pp. 487–511.

[13] C. Shorten and T. M. Khoshgoftaar, A survey on image data augmentation for deep learning, Journal of Big Data **6**, 60 (2019).

[14] Z. Cheng, X. Ren, F. Juefei-Xu, W. Xue, Q. Guo, L. Ma, and J. Zhao, Deepmix: Online auto data augmentation for robust visual object tracking, arXiv preprint arXiv:2104.11695 (2021).

[15] E. Poslavskaya and A. Korolev, Encoding categorical data: Is there yet anything 'hotter' than one-hot encoding? (2023), arXiv:2312.16930 [cs.LG].

[16] S. Ruder, An overview of gradient descent optimization algorithms (2017), arXiv:1609.04747 [cs.LG].

[17] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization (2017), arXiv:1412.6980 [cs.LG].

[18] SunFounder, Sunfounder picar-v smart car kit for raspberry pi (2024).

[19] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems (2015), software available from tensorflow.org.

[20] TensorFlow, Post-training quantization, `https://www.tensorflow.org/lite/performance/post_training_quantization` (2024), accessed: 2024-05-16.

[21] K. W. Bowyer, N. V. Chawla, L. O. Hall, and W. P. Kegelmeyer, SMOTE: synthetic minority over-sampling technique, CoRR **abs/1106.1813** (2011), 1106.1813.

[22] Y. Li, H. Mao, R. Girshick, and K. He, Exploring plain vision transformer backbones for object detection (2022), arXiv:2203.16527 [cs.CV].

[23] L. Wang, J. Liu, H. Shao, W. Wang, R. Chen, Y. Liu, and S. L. Waslander, Efficient reinforcement learning for autonomous driving with parameterized skills and priors (2023), arXiv:2305.04412 [cs.RO].