A DEEP LEARNING APPROACH TO AUTONOMOUS DRIVING

Oliver Wiech
University of Nottingham

ABSTRACT

In the past decade self-driving vehicles have become available to wide consumer market, mostly thanks to Tesla and Google which started research on self-driving cars in the year of 2009. Nowadays, numerous vehicle manufacturers actively research and build autonomous cars. The self-driving capabilities are achieved by developing models which guide the car travel. The deep learning models are capable of detecting objects on vehicle's travel path using in-built sensors. This work explores the topic to control Sun-Founder PiCar [1] with front-facing camera installed. Several 'from scratch' and pre-trained models' performance is evaluated using holdout set. The achieved results are thoroughly discussed, and advantages or disadvantages explained with their possible causes. The study uses several newly released techniques to get the top possible accuracy, such as the AdaBelief optimizer and EfficientNet architectures. Possible improvement areas are noted explaining the impact on the models developed.

Index Terms: AdaBelief, Autonomous Driving, Deep Learning, EfficientNet, Convolutional Neural Networks

I. INTRODUCTION

Neural networks are the core of every modern auto-pilot software. Tesla's software uses 48 networks that output a thousand of distinct tensors every time step to make a prediction of how the car should behave [2]. The study aims to research how deep learning can be used to operate an autonomous car equipped with front-facing camera. The images, with the resolution of 320x240 and RGB scale, are processed using extensive data augmentation and used as the input to Convolutional Neural Networks, neural architecture that extracts features from images and uses them to predict an output, in this case the car's angle and speed that should be set given the circumstances appeared on the image. The research goal is focused on two parts: Kaggle competition that uses the model to get predictions on unseen data and evaluate its performance, and live testing that uses autonomous car running on several tracks with different obstacles and real-world scenarios. The tracks, shown on Fig. 1, have three different shapes and include challenges such as stopping if a pedestrian is in the road or before red lights, performing a turn in response to traffic signs or ignoring objects outside the road, some shown on Fig 2. The challenge also assumes UK driving rules, that is driving on the left lane.

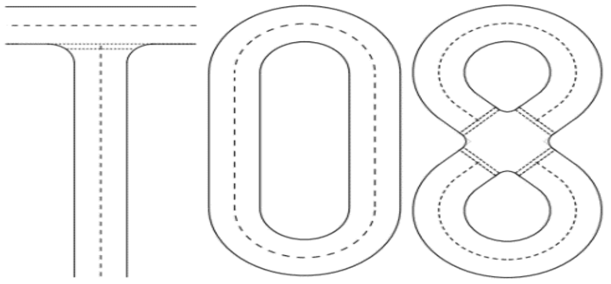
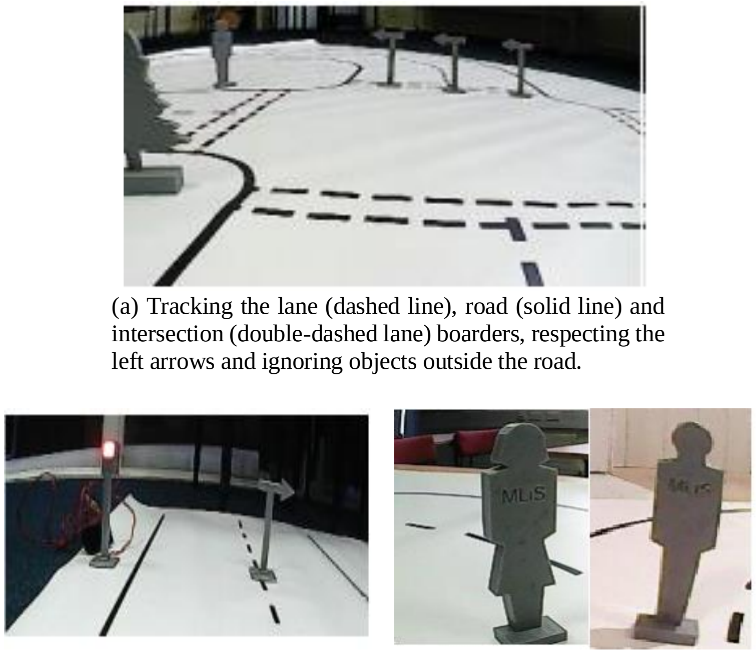


Figure 1 The three tracks that are used during the live testing: T-junction, oval and figure-of-eight.


(a) Tracking the lane (dashed line), road (solid line) and intersection (double-dashed lane) boarders, respecting the left arrows and ignoring objects outside the road.

(b) red lights, right turn arrow

(c) pedestrians on the road

Figure 2 Examples **of** challenges that appear on live-test track.

II. METHODS AND METHODOLOGY

A. PiCar and Target Values

Sun-Founder PiCar, the autonomous car used to test the model on live track, has a built-in front facing camera that is used to collect the images that are input into the model. The car is additionally powered by Raspberry Pi CPU with Coral Edge TPU, an edge computing device that greatly reduces the inference time of the model. The model, to use the TPU capabilities, had to be transformed into TensorFlow Lite version. The car is controlled by the outputs of the model which are equal to the speed and steering angle values. Both of them are normalized to have values between 0 and 1:

$$angle_{norm} = \frac{angle - 50}{80}$$
$$speed_{norm} = \frac{\text{speed} - 0}{35} \tag{1}$$

B. Dataset and Image Augmentation

The training data was generated by running the PiCar over the three tracks with various scenarios and objects on or near the track. The dataset consists of 13,000 train images that are labeled with the target responses of speed and steering angle, and 1020 test images. The training data was split into train and validation split with the ratio of 80/20. To improve the model training, the data was extensively augmented using similar approach to Lv et al [3]. Firstly, the RGB values are scaled from [0, 255] to [0, 1]. Then, using imgaug library specifically made for machine learning data augmentation, a custom sequential data augmentation object is created. The class randomly applies different transformations to the source image to increase the samples in the dataset. The transformations applied, shown on Fig. 3, include affine, contrast manipulation, motion blurring, perspective change, and adding extra objects. Adding extra objects required manual cropping and labeling the images, example from Fig. 4 shows adding a second person in front of the car, the added label sets the speed to zero.

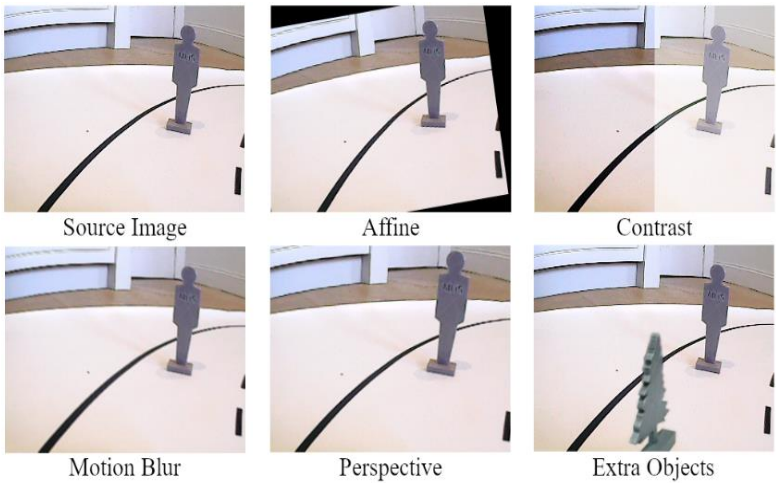


Figure 3 Examples **of** data augmentation applied to the training set.

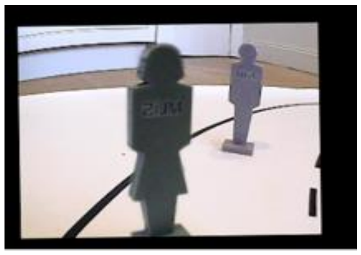


Figure 4 Adding extra objects to the source image required manual cropping and labeling.

Some of the images are also randomly mirrored. To keep the proper UK driving rules applied, the labels of the mirrored image must be changed by inverting the steering angle.

C. Deep Learning Architectures

'From Scratch' models, there are two neural network models that were written and trained from the scratch. The first one is a seven layers convolutional neural network that consists of two 'Conv-ReLU-Pool' blocks followed by the classification part of three dense layers with one dropout layer to increase generalization, the architecture is shown on Fig. 4. The second convolutional neural network takes inspiration from Inception Network and consists of seven inception blocks, that is a block that takes an input and process it in parallel manner through three types of convolution (1x1, 3x3, 5x5) and a max-pooling, the outputs of each is then concatenated to a final output from the block (Fig. 5). Inception block is a good approach to quickly find features of different level on the image, which leads to better accuracy and reducing training time. The architecture has a dropout and a dense layer with Sigmoid activation function as the two final layers.

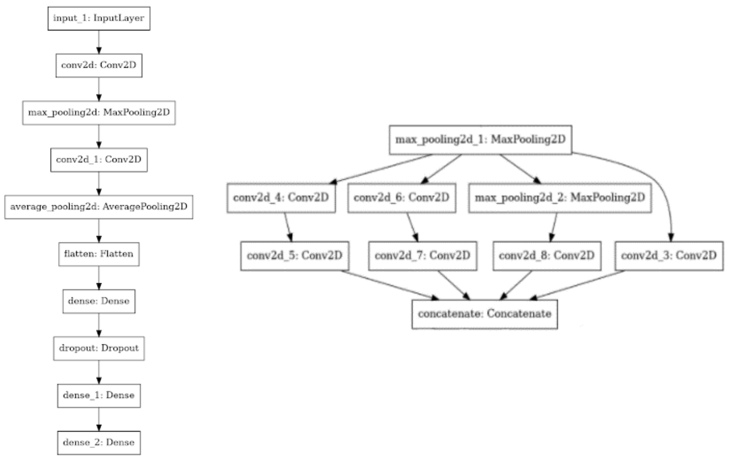


Figure 5 A simple CNN used in the study (left) and one inception block that is basis of the second model (right).

Transfer-learning models, there are several pre-trained models used in the project. The goal of transfer-learning is using weights that were pre-trained on a larger dataset in order to transfer the learnt knowledge to help with solving another problem. There are seven pre-trained models tested in this study, each of different properties and characteristics: very large deep networks ResNetV2, InceptionResNetV2, EfficientNetb6 and B2, as well as smaller and newer architectures of NASNetMobile, MobileNetV2 and EfficientNetB1. Each of them was pre-trained on ImageNet Large Scale Visual Recognition Dataset The main emphasis was put on the last-mentioned model, EfficientNetB1, due to its extraordinary results and new architecture that greatly improves the training process and the final accuracy compared to the older models.

EfficientNetB1, a model that belongs to a larger group of EfficientNet architectures that used many innovative features to increase the accuracy and the efficiency of the models. The crucial base of EfficientNets is compound uniform scaling [4]. It is known that scaling only depth in a neural network enhances the generalization and allows capturing more complex features at the cost of difficult training. On the other hand, scaling width makes neural nets easier to train and helps with capturing fine-grained features, similarly increasing resolution improves the results by giving more pixel information. The compound uniform scaling increases the depth, width, and resolution by a factor of b uniformly, combining the advantages of each dimension increase and negating the disadvantages of scaling only one of them separately. The base model, to be scaled, is found using Neural Architecture Search, a framework proposed by Google in [5] that uses Reinforcement Learning to find a proper architecture to a given problem using given building blocks. The base model is scaled with different parameter of b to create nine architectures of EfficientNet named from B0 to B8, each having around twice more number parameters than the previous one. The study focuses on B1 version that has the optimal accuracy to efficiency ratio, which is crucial to lower the inference time to be used on Raspberry Pi CPU. The EfficientNetB1 has 339 Layers with 7.8M of Parameters, around ten times less than ResNet152.

D. Optimiser and Hyperparameters

The project uses newly released AdaBelief optimizer, described in [6]. The optimization algorithm can be explained as:

$$\begin{aligned}
&\mathit{while}\ \theta_t\ \mathit{not\ converged}:\\
&t = t + 1\\
&g_t = \nabla_\theta f_t(\theta_{t-1})\\
&m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t\\
&s_t = \beta_2 s_{t-1} + (1-\beta_2)(g_t - m_t)^2
\end{aligned} \tag{2}$$

$$\begin{aligned}
&\mathit{bias\ correction}:\\
&\widehat{m_t} = \frac{m_t}{1-\beta_1^t}, \qquad \hat{s}_t = \frac{s_t + \in}{1-\beta_2^t}\\
&\mathit{update}:\\
&\theta_t = \prod \mathrm{F}, \sqrt{\hat{s}_t}\left(\theta_{t-1} - \frac{\alpha\widehat{m_t}}{\sqrt{\hat{s}_t} + \in}\right)
\end{aligned}$$

where $f(\theta)$ is loss function to optimize given weights θ, g_t is the gradient at timestep t, m_t is the exponential moving average of the gradient, β_1 and β_2 are the hyperparameters passed to the optimizer, which by default Keras sets to 0.9 and 0.999, α is the learning function and $\in$ is a very small number so denominator is other than zero. The AdaBelief optimiser is almost identical to Adam, it has the similar parameters, with one slight change when calculating the EMA of squared gradient: it has the addition of $(g_t - m_t)^2$ which is the square of the difference of the gradient and its exponential moving average. This allows for AdaBelief to take larger steps when the gradient is small, and small steps when the gradient is large which reduces the training time compared to using Adam optimizer. The learning rate used in this project is 1e-3 with Reduce Learning Rate On Plateau method added that decreases the learning rate by a factor of ten after two consecutive epochs of not improving the validation accuracy.

The weights used for pre-trained models were the default ImageNet weights from Keras [7]. EfficientNetB1, however, uses Noisy-Student weights, a framework which firstly trains 'teacher' model on labeled data that is then used for labeling new data. The new data is used for training 'student' model which becomes the teacher. The procedure is semi-supervised learning approach.

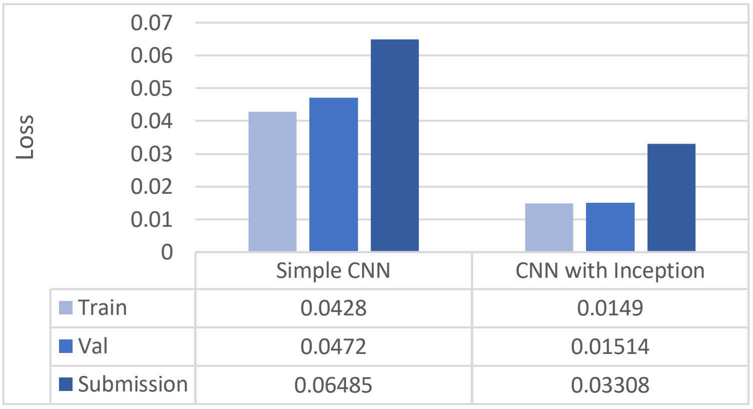
The classification part of the pre-trained model is changed to match the research problem and consist of two dense layers, first with 1024 neuron units and ReLU function and second with 2 units and Sigmoid activation. Freezing the layers in the feature extraction part of the model resulted in a mediocre accuracy results, therefore no layers were frozen so the weights and biases could be adjusted appropriately. The training lasts 30 epochs with Early Stopping introduced that stops the run if no improvement happens within 4 consecutive epochs. The batch size is set to 32, as proven empirically, and according to Masters et al [8], smaller batch sizes are preferable for faster training and result in better generalization of the model due to the noise introduced in small-batch data. The loss function used is the Mean Squared Error:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y_i})^2 \tag{3}$$

where $y_i - \tilde{y_i}$ is the difference between the predicted and target values.

III. RESULTS AND DISCUSSION

The results of 'from scratch' models, presented on Fig. 6, show that the simple CNN was not complex or deep enough for the task. Adding several inception modules allowed the model to better capture the fine-grained features on images and learn the complex patterns of the data and therefore greatly increased the accuracy. The submission score of CNN with inception is twice better than the simple CNN with just seven layers.



	Simple CNN	CNN with Inception
▨ Train	0.0428	0.0149
■ Val	0.0472	0.01514
■ Submission	0.06485	0.03308

Figure 6 Train, validation and submission results for simple CNN and inception-based CNN

Transfer learning, however, led to much lower loss and better accuracy than using 'from scratch' models. Table 1 shows the two coded CNN models compared to seven pre-trained models.

Model	Train	Validation	Submission
Simple CNN	0.0428	0.0472	0.06485
CNN with Inception	0.0149	0.01514	0.03308
NASNetMobile	0.0076	0.00787	0.01414
EfficientNetB6	0.008	0.0089	-
MobileNetV2	0.0093	0.00945	-
ResNetV2	0.0127	0.0117	-
InceptionResNetV2	0.025	0.01115	-
EfficientNetB2	0.00982	0.00772	-
EfficientNetB1	0.0078	0.00788	0.01372

Table 1 Table with 'from scratch' (top 2 rows) and pre-trained models and their losses.

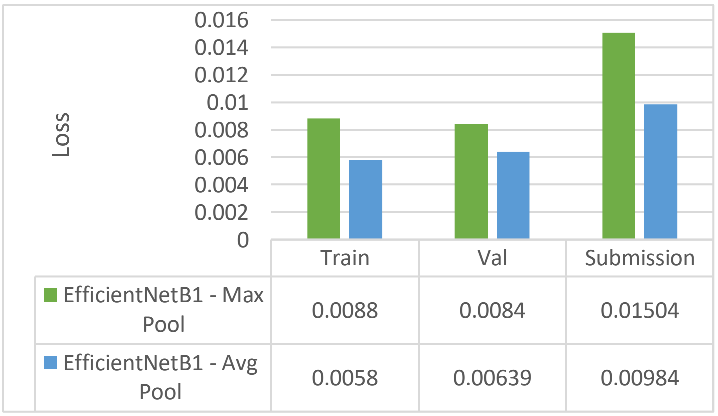
Even the worst performing pre-trained model achieved significantly better results on validation set than CNN with Inception blocks. The investigation on our study shows that lightweight and newer architectures (MobileNetV2, EfficientNetB1) perform better than the older and much larger models (ResNetV2, InceptionResNetV2). There are several reasons for that, and the differences why the newer models are better will be explained on EfficientNetB1 which was picked to be the final model in our study due to its unmatched performance.

EfficientNetB1, compared to old architectures, has a lot of further enhancements that improve its score. The first notable one is using Swish-1 Activation function:

$$swish(x) = x * sigmoid(x) \qquad (4)$$

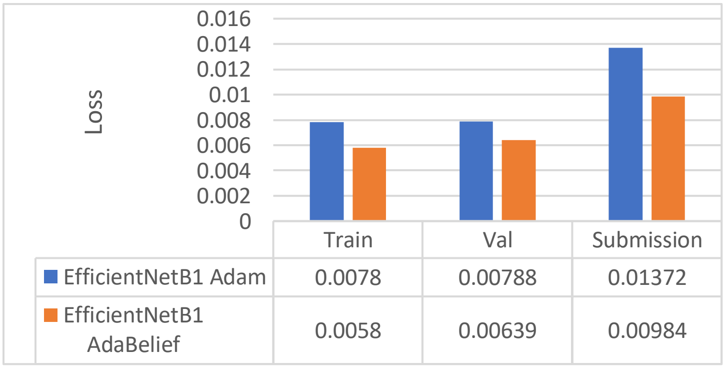
that is proven to increase the accuracy of the model compared to ReLU. Studies show that simply using Swish-1 instead of ReLU increased the ImageNet accuracy by 0.06 [9] due to its smoothness, especially with very deep nets above 42 layers where the optimization becomes difficult. The other noteworthy improvement is stochastic depth used during training. Stochastic depth shortens the network during training by randomly dropping blocks of layers and bypassing them with identity function, in case of EfficientNetB1 $P_{survival} = 0.8$. This combats the vanishing gradient problem and allows the error signal to better propagate through the entire network leading to better accuracy.

Keras allows to decide whether the imported pre-trained model should use max-pooling or average-pooling operations. As shown on Fig. 7 the results are very different. According to basic intuition, and proven by Zhou et al [10], average pooling suits the project's data better because max-pooling provides translation invariance, which is unwanted here since placements of objects matter for autonomous driving (a box outside the road boundary which should be ignored versus a box in the middle of the road).



	Train	Val	Submission
🟩 EfficientNetB1 - Max Pool	0.0088	0.0084	0.01504
🟦 EfficientNetB1 - Avg Pool	0.0058	0.00639	0.00984

Figure 7 The difference between using Max-Pooling and Average-Pooling in EfficientNetB1



	Train	Val	Submission
🟦 EfficientNetB1 Adam	0.0078	0.00788	0.01372
🟧 EfficientNetB1 AdaBelief	0.0058	0.00639	0.00984

Figure 8 AdaBelief and Adam difference in loss. AdaBelief converged after 14 epochs, while Adam needed 16.

As mentioned earlier, the project uses AdaBelief optimiser. The results, shown on Fig. 8, present the difference between AdaBelief and Adam. The former achieves better accuracy with faster convergence (14 vs 16 epochs) making it better overall optimiser for the task.

The final iteration of EfficientNetB1 uses, among others, AdaBelief, 32 batch-size and average-pooling. The training of this configuration is shown on Fig. 9. Both the training and validation losses converge relatively quickly to values around 0.005-0.006.

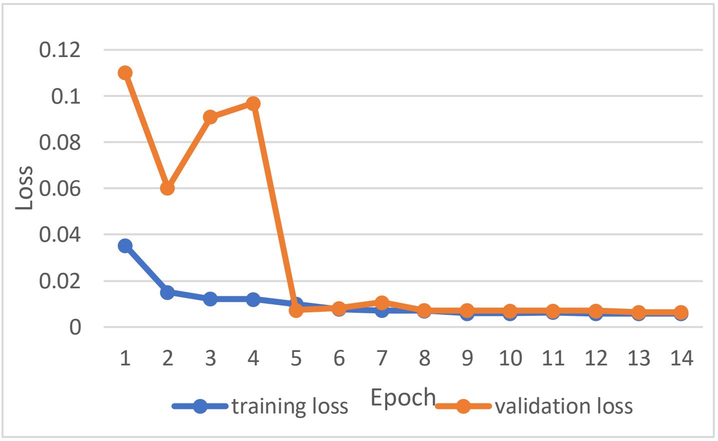
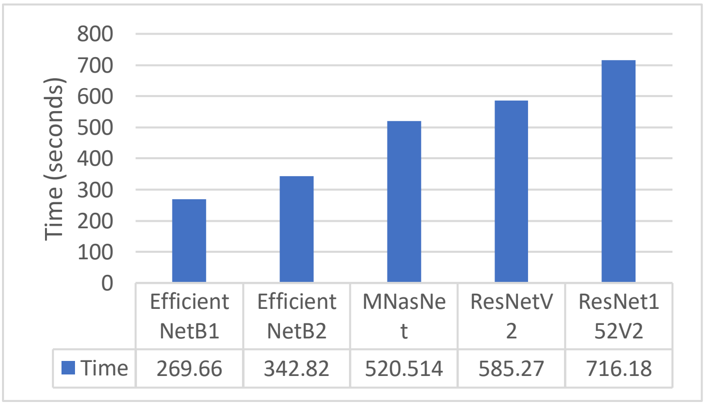


Figure 9 The best configuration of EfficientNetB1 training loss.

The resulting submission to Kaggle achieved second place in public leaderboard (0.00969 score) and first place in private leaderboard (0.00928).



	EfficientNetB1	EfficientNetB2	MNasNet	ResNetV2	ResNet152V2
■ Time	269.66	342.82	520.514	585.27	716.18

Figure 10 The test set (1020 images) evaluation times (Colab CPU/no accelerator).

For autonomous driving, the inference time of the model is crucial. Figure 10 shows the evaluation of test dataset using Google Colab CPU with no additional accelerator. EfficientNetB1 is significantly faster than the competitor models due to its efficiency optimization techniques used: MBConv Blocks (Inverted-Residual-Blocks) that are much faster than normal Convolutions, and squeeze-and-excitation which dynamically recalibrate channel-wise features. All of this leads to low inference time and mobile-friendly model.

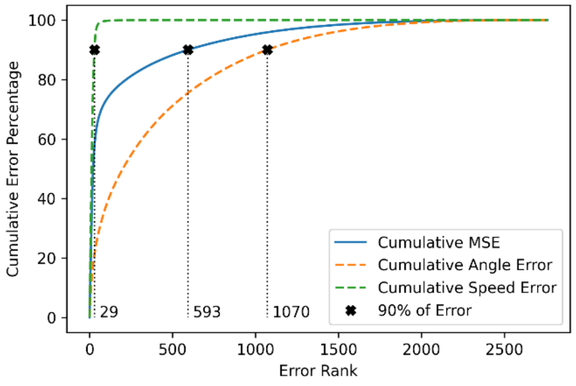


Figure 11 Distribution of errors using EfficientNetB1.

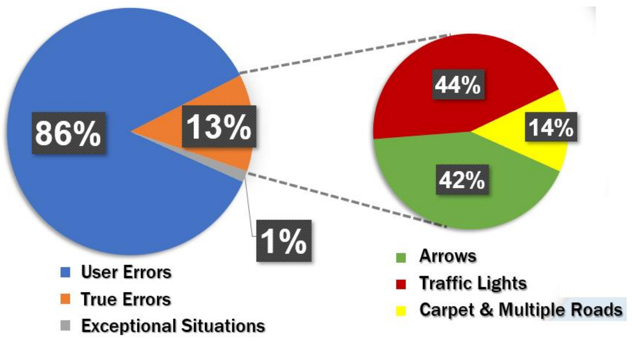
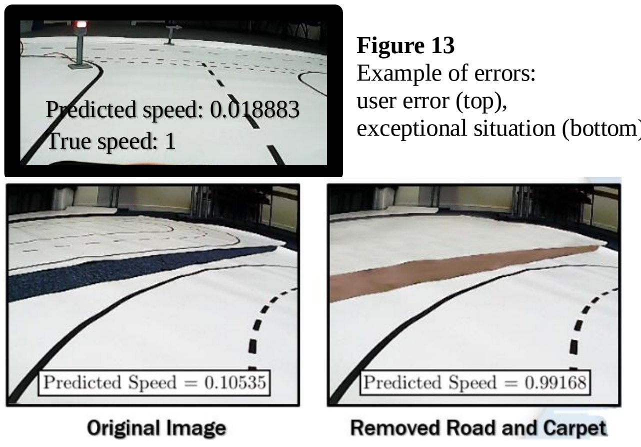


Figure 12 Top 50 Mean Squared Error contributors.

Fig. 11 shows the distributions of errors in validation set. While difference in angle speeds is expected, it is useful to see what causes the speed errors. Fig. 12 shows the origins of the errors: caused by exceptional situations (1%), inappropriate user control during gathering data (86%) and only 13% of the errors are true model errors that were mostly caused by traffic lights and road arrows. Example on Fig. 13 shows that the image has labeled with speed of 1 when seeing the red lights, and it also shows the background noise: seeing second track. Such data noise, user errors and wrong labels make the model train incorrectly reducing its performance.



Predicted speed: 0.018883
True speed: 1
Predicted Speed = 0.10535
Predicted Speed = 0.99168
Original Image
Removed Road and Carpet

Figure 13 Example of errors: user error (top), exceptional situation (bottom)

The live testing was a success, only two mistakes were made: crossing red-lights and wrong turn on left-turn arrows, shown on Fig. 14.

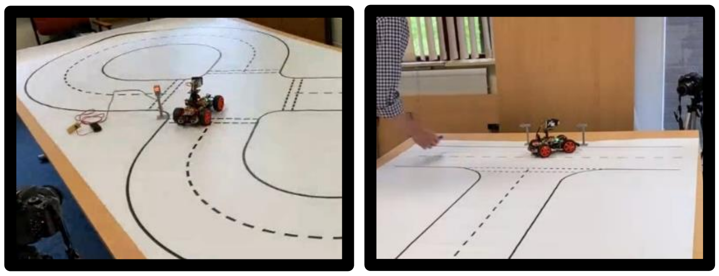


Figure 14 Live testing errors: passing through the red lights and turning right on left-arrows.

It is difficult to tell what caused the errors without seeing the camera output. The red lights could have not be seen by the front-facing camera, or there might have been a background noise causing the error. It is noteworthy that there is no temporal memory, just when the cars stop seeing the red lights, it will act like they were never there. The arrow error might be due to imbalanced data with very few left-turn arrows. To solve this, further data augmentation with the arrows should have been done. However, it was impressive how well the model generalizes to real-life scenarios and successfully completes all of the other challenges, resulting in 1st place achieved not only on Kaggle but also on the live testing tracks.

IV. CONCLUSION

A. Improvements

There are many improvements that could not have been done due to the time constraints. The first one is using traditional Computer Vision methods to detect lines (Hough Transform, edge detectors) and then use that knowledge to reduce the background noise influence on the model. AutoAugmentation method could also be applied to improve neural network validation accuracy by improving data augmentation even further in an automated manner. Object detection algorithms could be added to make the model more robust, and manual code added for dealing with traffic lights by emulating temporal dynamics: taking in mind when the red lights started and not running till green lights. The final suggested improvement would be going over the dataset and manually rechecking the labels to reduce the effect of user errors mentioned earlier.

B. Conclusion

The study demonstrated successful way of using deep learning models to control autonomous vehicles. Convolutional Neural Networks architectures, especially pre-trained large models, are excellent tool for extracting relevant information from car's camera output and processing the features to give predicted speed and angle that the car should set in given situation. A single model was enough to achieve great score on testing set in Kaggle and to complete most of the task in live testing. Having said that, in a true real-life scenario with a normal car it would not be sufficient due to much larger number of objects and challenges appearing on roads. Most likely, tens of networks would be needed to work together as an intelligent agent by processing different features and multiple sensors' data, similarly to Tesla's approach, to be robust enough to operate in real-world cities and roads. Nevertheless, seeing current research and improvements in autonomous driving there is no doubt that deep learning is, and will be, a crucial part of every self-driving car in the foreseeable future, most likely leading to level five in the Society of Automotive Engineers scale of autonomous vehicles which means full automation with no human driver input needed at any point at all.

REFERENCES

[1] SunFounder PiCar-S Kit V2.0 for Raspberry Pi, https://www.sunfounder.com/products/raspberrypi-sensor-car, [accessed 16th May 2021]

[2] S. Loveday, 'What's Behind Tesla's Neural Network For Autopilot And Self-Driving?', https://insideevs.com/news/396126/tesla-autopilot-neural-network-advancements/, [accessed 16th May 2021]

[3] J. Lv, X. Shao, J. Huang, X. Zhou. 'Data augmentation for face recognition', Neurocomputing, Vol. 230, p. 184-196, 22 March 2017

[4] M. Tan, Q. Le. 'EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks', arXiv:1905.11946, ICML 2019

[5] B. Zoph, Q. Le. 'Neural Architecture Search with Reinforcement Learning', arXiv:1611.01578, 2017

[6] J. Zhuang et al, 'AdaBelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients', arXiv: 2010.07468v5, 2020

[7] Y. Fu. 'Image classification via fine-tuning with EfficientNet', https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/, [accessed 16th May 2021]

[8] D. Masters, C. Luschi. 'Revisiting small batch training for deep neural networks', arXiv: 1804.07612v1, 2018

[9] P. Ramachandran, B. Zoph, Q. Le. 'Searching for activation functions', arXiv: 1710.05941v2, 2017

[10] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, A. Torralba. 'Learning Deep Features for Discriminative Localization', arXiv: arXiv:1512.04150v1, 2015