MLIS 2: Autonomous Driving Project

Tom Roper

In this project, we used convolutional neural networks (CNNs) to autonomously drive a toy car in an end-end framework. We found that pretrained DenseNet architectures, with custom extra layers, performed better than other pretrained models. Out of our cohort, we scored 2nd and 1st in prediction accuracy and driving ability, respectively. Our model completed all driving manoeuvres apart from turning left at a junction. In future work, to improve this, we would explore CNN+LSTM, object detection and visual transformer models. Furthermore, we will explore the generalisability of our models to more realistic environments such as the CARLA driving simulator.

1. Introduction

Autonomous driving can be defined as teaching vehicles to drive automatically (i.e. to a specified destination) with minimal or zero human interaction. This requires taking sensory input from the vehicle, like a camera, and converting it into driving decisions (like steering wheel angles or acceleration commands). Research into autonomous driving could greatly improve driver safety, as some estimates show that 94% of crashes are due to human error [1]. Provided that we can develop driving models that exceed human capabilities, which is starting to happen in some domains [2], this presents an opportunity to make a meaningfully positive impact to human safety. One solution to autonomous driving is to create modules that handle each task, separating the visual perception, navigation and driving systems. However, work in 1988 demonstrated that it was possible to learn the entire autonomous driving pipeline with just one 3-layered neural network model [3]. Their relatively simple model could learn from low resolution grayscale images and follow roads under fixed environment conditions. This approach of mapping raw sensory input to final decisions is called end-end learning (Fig. 1) and is commonly adopted in autonomous driving.

Recent advances in computer power and deep learning algorithms, like convolutional neural networks (CNNs), have revolutionised research in end-end driving systems. NVIDIA were early to implement a CNN driving model, which could accurately steer across a variety of road types (local roads, highways, parking lots), from only 72 hours of training data [5]. We contribute to the field of autonomous driving by using CNNs to predict steering angles and speeds of a toy car. Our model performance was benchmarked against our university peers, for prediction accuracy (via a Kaggle competition) and for its driving ability (live testing). We used CNNs due to their prevalence in the autonomous driving literature and abundance of pretrained models available to transfer learn for our task. The paper begins with our methodology, then covers the results of the Kaggle competition and live testing. Finally, we end on a discussion of potential improvements and future directions.

2. Methods

2.1 Test Vehicle – Sunfounder Pi-Car

We trained the Sunfounder Pi-Car [6] to autonomously drive, equipped with raspberry pi, TensorFlow and an optional TPU. The on-board camera can send a live stream of images directly to its processing units for use in prediction. The predicted angles and speeds of our model are then sent to the vehicle actuators to control movement.

2.2 Dataset

We were provided with 13,800 training images taken as the car was driving around the tracks and completing different driving manoeuvrers (Fig. 2). These included, stopping for pedestrians or red traffic lights and turning at junctions, all while in the left-hand lane.

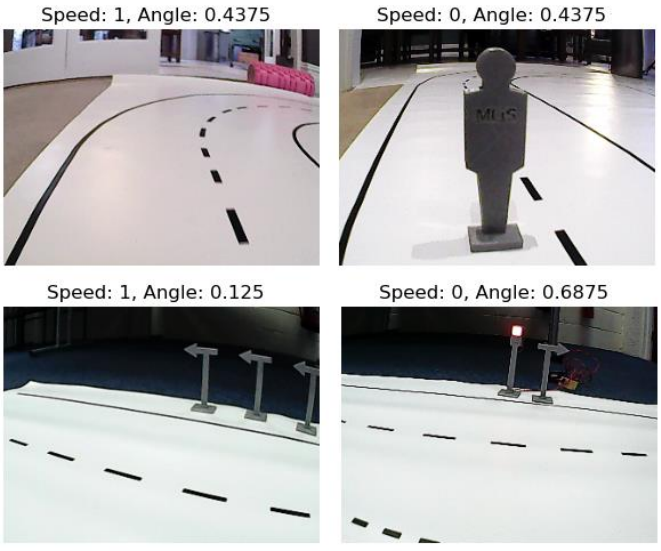

Figure 2. Four sample images from the provided dataset. Supervision labels show correct angle and speed for each track location

The images are labelled with angles and speeds that form the supervision for our CNN modelling. Angle labels had 17 categories between 0 and 1, with 0.5

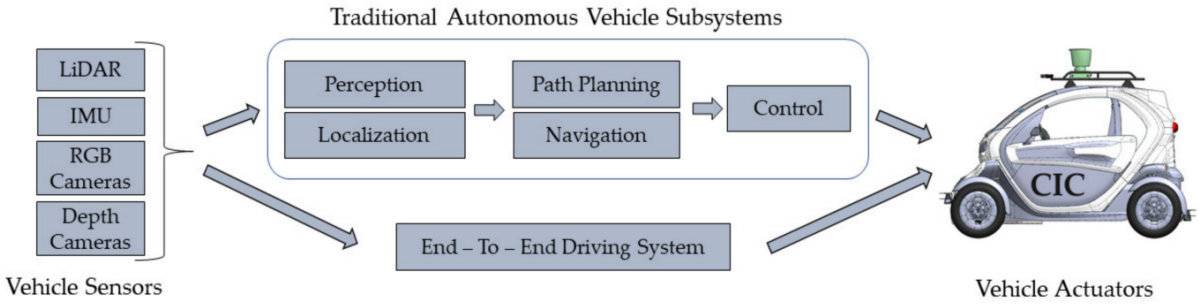

Figure 1. Flow diagram highlighting difference between modular and end-end autonomous driving systems. Taken from [4].

corresponding to driving straight, angles above 0.5 for turning right and below 0.5 for turning left. Speed is either 0 or 1, corresponding to stopping or driving respectively. All driving behaviour takes place on three different tracks: The oval, figure of 8 and T-junction (Fig. 3).

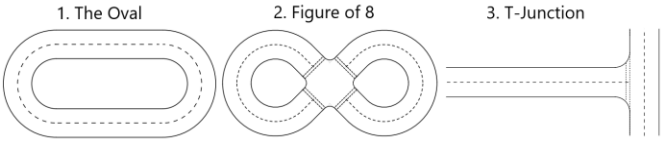

Figure 3. The three driving tracks. Tracks were used for live testing and are where training data was captured.

2.3 Data Preprocessing

We found 5 corrupted images and 1 abnormal speed value (greater than 1), which we removed, leaving us with 13,794 images in total. There was an imbalance in both target variables, as 75% of the images have a speed equal to 1, and there is a bias for right-hand turns (Fig. 4). We tested weighting the loss function more heavily or providing more data augmentation to the underrepresented classes. However, weighting the loss function did not significantly improve model performance, and augmenting the data equally, irrespective of label class, had best performance outcomes. We discuss our performance guided approach to model selection and optimization in section 2.5.

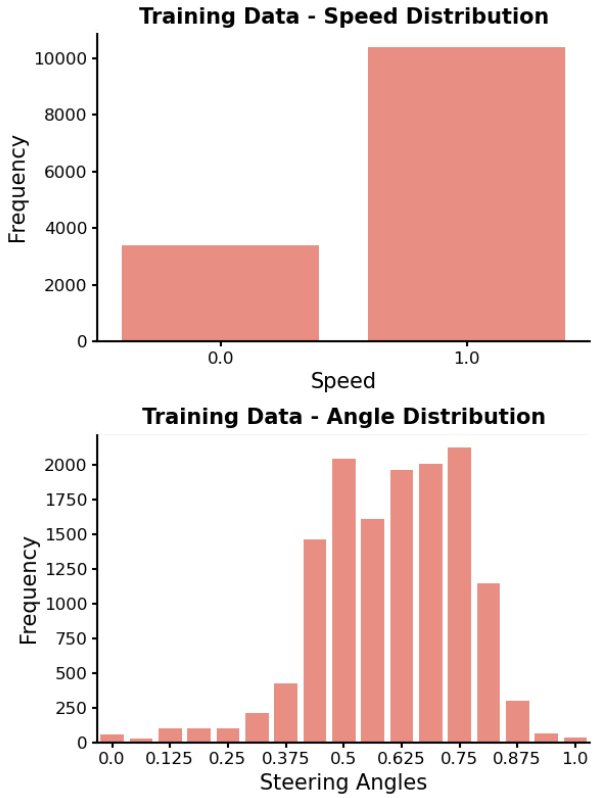

Figure 4. Both target variables show imbalance. Within the training dataset, there is a bias for speed = 1 and rightward turning angles.

2.4 Data Augmentation

We used data augmentation to increase the data we had and increase the robustness of our models. Blurring the images can remove tiny details of the images that our models might be overfitting to, as demonstrated previously [7]. Flipping the images horizontally also removed the bias for right-hand turns (Fig. 5).

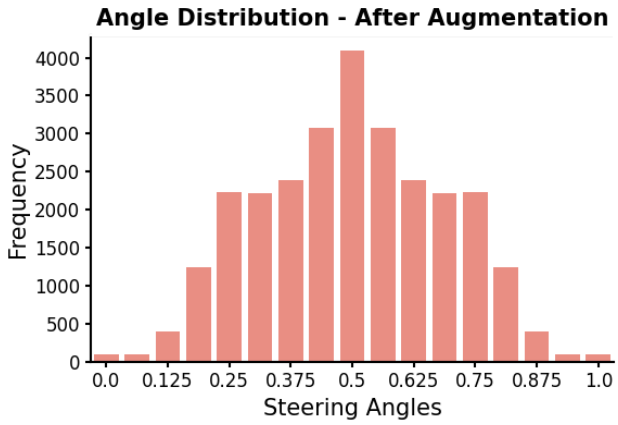

Figure 5. Horizontal flipping data removes turning bias.

We found best results by taking each original image and creating a single augmented copy of it, with all the 4 following modifications applied: Random blur, contrast and brightness and a horizontal flip (Fig. 6).

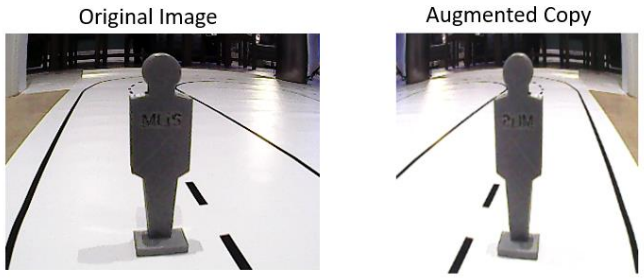

Figure 6. Example of original image (left) and image after augmentation (right). Random brightness had a max delta of 0.2, contrast and saturation were random within a range of 0.8-1.2. After horizontal flipping we changed the angle label to be 1 minus the original angle. We resized all images to be of shape 224x224 (shown in right image), as required by our model architecture, and normalised them to have pixel values between 0 and 1.

After data augmentation we had 27,584 training images, randomly shuffled and split into 80% for training data and 20% for validation data. Random shuffling ensures the data is randomly distributed between validation and training sets. Without shuffling, the validation set might only be comprised of augmented images which would not allow a fair test of validating our model performance.

2.5 Model Selection and Optimization

Mainly, we used the mean-squared error (MSE) on our validation data to guide our decisions:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left[(g_i - \hat{g}_i)^2 + (h_i - \hat{h}_i)^2\right] \quad (1)$$

Where n is the total data points, g_i the real angle label, h_i the real speed label and $\hat{g}_i/\hat{h}_i$ are the predicted angles and speeds, respectively, all for the ith datapoint. We would test different modelling choices (pretrained architectures, hyperparameters, augmentation types, etc) and focus on whichever had the lowest validation MSE. However, in the live-testing section, we did place more consideration on the driving ability of the car. For the Kaggle and live testing models we found the same training hyperparameters to perform best (Table 1).

Hyperparameter	Value/Function
Batch size	32
Optimizer	Adam
Epochs	100
Loss Function	MSE
Early stopping	If validation loss does not improve within 10 epochs.
Trainable Layers	All unfrozen: Pre-trained and new layers
Learning rate	0.0001

Table 1: HyperParameter settings for model training

3. Results

3.1 Kaggle Competition

For the competition, we had 43 days to create a model that scored the lowest MSE on predictions of a test dataset, meaning that only prediction accuracy mattered. The test data was 2040 images taken from the same distribution as our training data. Half was available to test our model against (once per day), and half was held-back to produce final scores at competition end.

Overall, we made 18 submissions, with our best performing model being a pretrained DenseNet-169 [8] with custom top layer structure (Fig. 7). We also tried ResNet50V2 [9], VGG19 [10] and InceptionV3 [11] models, but all had higher validation and testing MSE. Additional to the pretrained model, we appended two shared convolutional layers (256 filters at 1x1 followed by 128 at 3x3), followed by a branch that separates the angle and speed predictions, taking inspiration from [4]. Each branch is identical, with an additional convolutional layer (64 filters at 1x1), followed by a fully connected layer with drop-out (0.3) and linear activations for the output node (clipped to be between 0 and 1), treated as a regression problem given the use of MSE in testing. Convolutions are followed by rectified linear unit (ReLU) functions and there is a global average pooling before the fully connected layer. DenseNet is unique because it has *Dense blocks* that permit increased information flow through the network. This helps with the vanishing gradient problem, and it outperforms networks like ResNet with equal

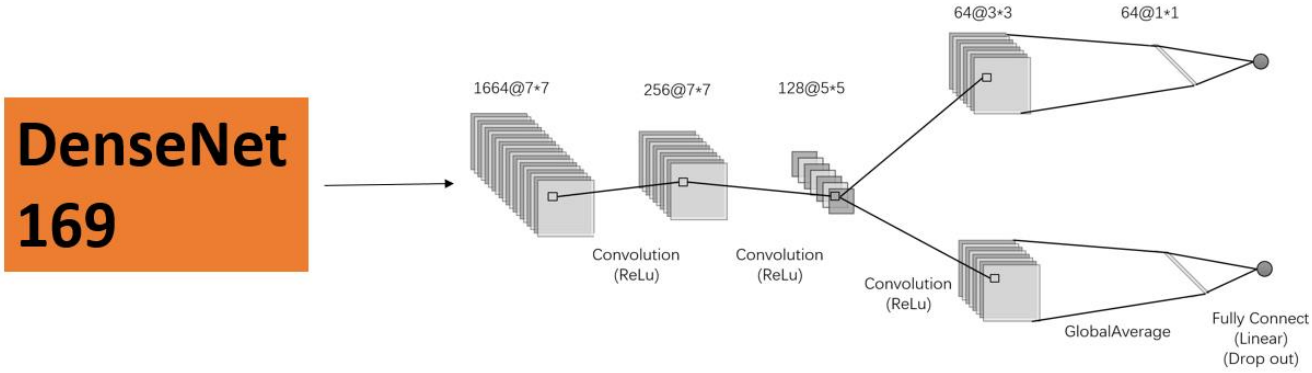

Figure 7. Kaggle model DenseNet structure. Trainable parameters = 13,353,986.

parameters numbers [8]. Dense blocks show similarity to ResNet skip layers [9], where the input to the layers can skip the convolution operation and be directly summed with feature maps of another layer. However, in Dense blocks, the inputs (feature maps) that skip to other layers are concatenated, not summed, to the output feature maps of the connected layers. Additionally, there are direct skip connections to all layers that share the same feature map size, rather than just a single skip connection. This aids the flow of gradients during backpropagation, promotes the reuse of feature maps to reduce redundant layers and provides a source of implicit deep supervision that makes learning more effective. After 42 epochs of training, our final model had a validation MSE of 0.0106, with 0.0046 for angle and 0.006 for speed (Fig. 8).

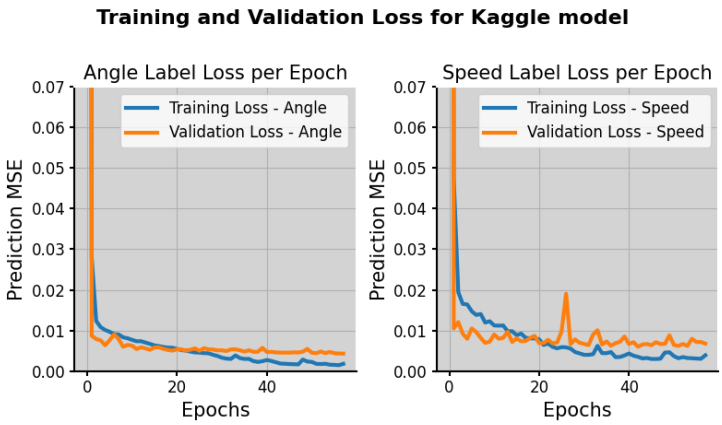

Figure 8. Training and Validation MSE per epoch for final Kaggle model

Our final testing MSE was 0.0101, similar to our validation MSE of 0.0106, indicating minimal overfitting. This put our team (ToFang) in 2nd place out of 16 total teams (Table 2).

Position	Team Name	Submissions	MSE Score
1	The autopilot	25	0.0086
2	**ToFang**	**18**	**0.0101**
3	RasPi Gosling	11	0.0111
4	SPC 1718	22	0.0129
5	2 Men 1 Car	32	0.0133

Table 2: Kaggle Leaderboard (Top 5)

3.2 Live Testing

In live testing, our model was tested in its ability to drive the car around 12 different scenarios (Fig. 9). Each was scored from 0-3 with 3 indicating perfect performance and 0 indicating complete failure.

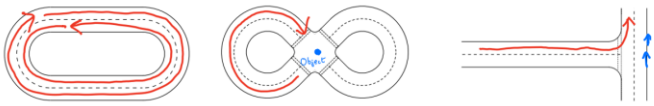

Figure 9. Examples live testing scenarios. Left: Driving round oval track. Middle: Driving round figure 8 and stopping at intersection because of object. Right: Turning left at T-junction based on turning sign direction.

Driving in real-time meant that model processing speed was now an important consideration, and it was recommended not to exceed 300ms of inference time. We tested our Kaggle model on the Pi-car, however it had an inference time of 2s per step. It could drive within lane markings in a straight line, but when it reached a corner it would not turn in time before it would drive off the track. We, therefore, explored solutions to reduce the inference time of our model without compromising its prediction accuracy. We tested MobileNetV3-Large [12] and DenseNet-121 [8], using the same additional layer structure as the Kaggle model. We used a MobileNet architecture due to its computationally efficient design, including depth-wise separable convolutions, which reduce the number of parameters and computational steps in the normal

convolution process. It also includes other features to enhance efficiency like bottleneck layers, inverse residuals and squeeze and excite modules. DenseNet-121 is a smaller model than the 169 version, with its dense blocks having overall 24 fewer layers. The models could complete more of the turn than before, but still drove off the track. The MobileNetV3 model also produced glitchy driving behaviour, driving along the straight track but by constantly switching between left and right turning directions. We therefore focused on DenseNet-121, which had improved our inference time to 800ms, through three final changes. We first decided to resize the input images to be of 112x112 dimension instead of 224x224 that we used for the Kaggle model (Fig. 10), taking inspiration from [13].

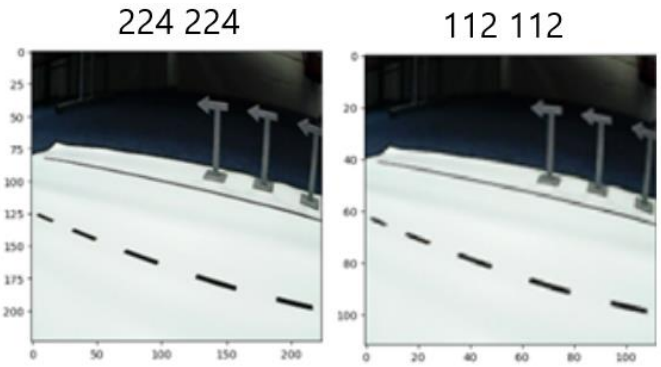

Figure 10. Resizing image to 112x112 still captures important details. Left is image size used for Kaggle model, right for live-testing model.

We then converted our model file to the tf-lite format [14], reducing the size of the file and improving its processing speed. This happens through: Quantising the model's weights (converting from float to integer), clustering similar weights to reduce the number of unique values and pruning neurones that contribute little to the output of the model. Tf-lite files can utilise the TPU present on the pi-car, but we found our model ran less smoothly on the TPU. Finally, we simplified the extra customer layers we added, lowering filter numbers in the first shared convolutional layer to 128 and removing the second shared convolutional layer. Now our inference time was 300ms per step. This model formed our final live testing model (Fig. 11), after 18 epochs of training it had a validation MSE of 0.0126, with 0.0057 for angle and 0.0069 for speed (Fig. 12).

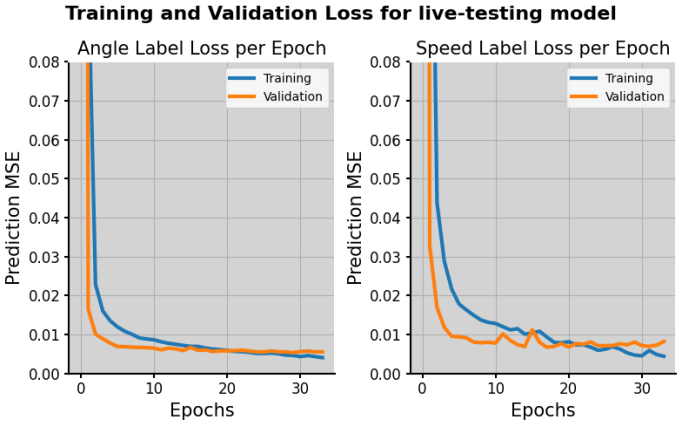

Figure 12. Training and validation MSE per epoch for final live testing model.

We tested the model on all driving scenarios expected in the live test and it completed all of them apart from the left-hand T-junction turn. These findings were replicated in the actual live testing of our model (Table 3), where we placed first out of our module cohort, scoring 33/36.

Position	Team Name	Driving Score
1	**ToFang**	**33**
2	Trojan Horses	29
3	XJJ	28
4	404: Driver not found	27
5	RASPI Gosling	27

Table 3: Live-Testing Leaderboard (Top 5)

Our single failed manoeuvre (Fig. 13) was failed by all teams, apart from the 'Autopilot' team.

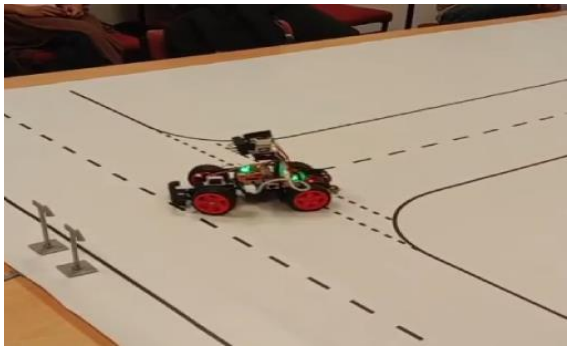

Figure 13. Single failed task – Left hand turn at T-junction. After reaching the T-junction, the car is supposed to make a left-hand turn according to the direction of the turning signals. Our model would always turn right at this junction, regardless of the sign direction.

To understand why our model failed this scenario we analysed the predictions of this manoeuvre using LIME explanations [15]. This involves systematically perturbing portions of the image and seeing how each

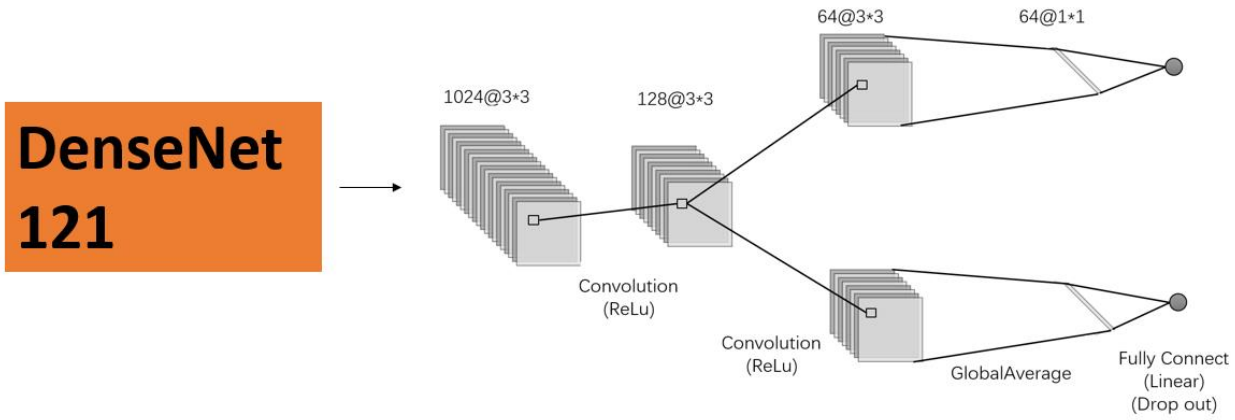

Figure 11. Final live testing model architecture. Trainable parameters = 7,659,522.

perturbation changes the prediction compared to the baseline of the normal image (Fig. 14).

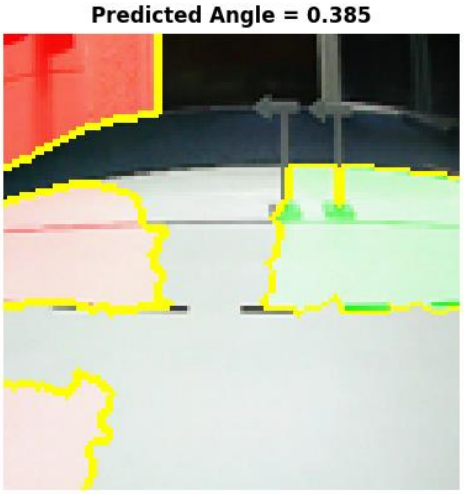

Figure 14. LIME explanation of angle prediction by live testing model on failed test. Red regions indicate parts of the image that are important for turning left, and green for turning right.

Despite failing this manoeuvre in testing, our model correctly predicted in this image to start turning left (angle=0.385). However, the region of the image most important (darkest red) for this decision was unrelated to the track. Furthermore, the turning signals were completely unimportant for the decision to turn left, indicating that the model had not learnt to detect these objects and use them for the prediction.

4. Discussion

Overall, our DenseNet architectures performed well as end-end driving models for prediction accuracy of angle and speed and the driving scenarios tested. To improve, we would like to complete the left-hand T-junction turn, which might be helped by collecting more data on the procedure. To label additional images, we could use a semi-supervised learning approach [16], where we use the Autopilot team's model to predict the angle and speed labels of the new images, as they could complete the manoeuvre.

We could also explore alternative deep learning architectures to see if they get better MSE and driving ability than CNNs. Previous work has combined CNN structures with long short-term memory (LSTM) layers, which are recurrent and can help the driving network to remember temporal information [17]. They demonstrate that the network can perform complex tasks like turning around at a dead end, which the CNN only architecture couldn't perform. They suggest that the LSTM layer gives access to temporal information, important for manoeuvres that rely on a sequence of events, which might help with the T-junction turn. One potential reason for failing the left-hand turn is not detecting the turning signals (Fig. 14). Therefore, we could explore object detection CNN models like *you only look once* (YOLO) [18], that predict coordinates of bounding boxes, and the object within them, in the same process. There is also a fast version [18] that uses 9 convolutional layers instead of 25 and achieves <25ms of inference time, which seems of relevance for our live testing scenario. To integrate this model into our end-end learning approach we could create a modified CNN architecture that processes the image in one branch, the YOLO bounding box and class predictions in another branch and concatenates them before the final fully connected layers of the CNN, similar to [4]. Object detection might help the model to identify the turning signs as important in making the junction turn manoeuvre.

We could also explore vision transformers [19], which process images with help of attentional components, first introduced in transformer architectures [20]. For steering angle prediction, visual transformers can

outperform CNNs and CNN+LSTM models on real driving images [21]. Importantly, this was at similar frames per second of processing time to the CNN network, so might not damage our inference time. We could also explore how our models generalise to realistic driving environments, testing whether they have learnt the underlying principles of driving or rely on uncommon road features like the white road colour and black road markings. We could test predicting the correct angle labels from the Sully Chen dataset, of real-life driving images from vehicles as they are driving around California [22] (Fig. 15). Despite the data being taken from a right-hand side perspective, our horizontal flip augmentation should have made our model comfortable with style of lane driving.

Figure 15. Two test images from the Sully Chen dataset.

We could also test our models on the CARLA driving simulator [23]. The simulator would feed a camera input to our model (Fig. 16) that predicts the steering angle of the car and whether to accelerate or brake. We could make our model causes acceleration (up to a maximum speed) if speed prediction is 1 and braking otherwise. We can then visually monitor how the model performs in simple scenarios like driving in the road or handling corners.

Figure 16. Sensory input of the simulated car from CARLA.

5. Conclusion

In this project, we developed and evaluated autonomous driving models for predicting angle and speed labels and navigating a driving track. Using custom pre-trained DenseNet architectures, we achieved high prediction accuracy and strong driving performance across two benchmarks. Future work will explore state-of-the-art architectures to further enhance performance and test the generalizability of our models in more realistic environments, such as real-world images and industry-standard driving simulators.

6. References

[1] Singh, S. (2015). Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey (No. DOT HS 812 115).

[2] Jones, N. (2024, April 15). AI now beats humans at basic tasks — new benchmarks are needed, says major report. *Nature*. https://doi.org/10.1038/d41586-024-01087-4

[3] Pomerleau, D. A. (1988). Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, *1*.

[4] Kocić, J., Jovičić, N., & Drndarević, V. (2019). An End-to-End Deep Neural Network for Autonomous Driving Designed for Embedded Automotive Platforms. *Sensors (Basel, Switzerland)*, *19*(9), 2064. https://doi.org/10.3390/s19092064

[5] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., & Zieba, K. (2016). End to end learning for self-driving cars. arXiv. https://arxiv.org/abs/1604.07316

[6] SunFounder. (n.d.). Smart Video Car. SunFounder. Retrieved May 17, 2024, from https://www.sunfounder.com/products/smart-video-car

[7] Gedraite, Estevao & Hadad, M.. (2011). Investigation on the effect of a Gaussian Blur in image filtering and segmentation. 393-396.

[8] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2016). Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*. Retrieved from https://arxiv.org/abs/1608.06993

[9] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*. Retrieved from https://arxiv.org/pdf/1512.03385

[10] Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*. Retrieved from https://arxiv.org/pdf/1409.1556

[11] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the inception architecture for computer vision. *arXiv preprint arXiv:1512.00567*. Retrieved from https://arxiv.org/pdf/1512.00567

[12] Howard, A., Sandler, M., Chu, G., Chen, L. C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., & Adam, H. (2019). Searching for MobileNetV3. *arXiv preprint arXiv:1905.02244*. Retrieved from https://arxiv.org/pdf/1905.02244

[13] Jang, W., Jeong, H., Kang, K., Dutt, N., & Kim, J.-C. (2020). R-TOD: Real-time object detector with minimized end-to-end delay for autonomous driving. *arXiv preprint arXiv:2011.06372*. Retrieved from https://arxiv.org/pdf/2011.06372

[14] TensorFlow Lite. (n.d.). *TensorFlow Lite: Lightweight solution for mobile and embedded devices*. TensorFlow. Retrieved from https://www.tensorflow.org/lite

[15] Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why should I trust you?": Explaining the predictions of any classifier. *arXiv preprint arXiv:1602.04938*. Retrieved from https://arxiv.org/pdf/1602.04938

[16] Chapelle, O., Scholkopf, B., & Zien, A. (Eds.). (2010). *Semi-supervised learning*. MIT Press. Retrieved from https://www.molgen.mpg.de/3659531/MITPress--SemiSupervised-Learning.pdf

[17] Lai, Z., & Braunl, T. (2023). End-to-end learning with memory models for complex autonomous driving tasks in indoor environments. *Journal of Intelligent & Robotic Systems, 105*(3), Article 42. https://doi.org/10.1007/s10846-022-01801-2

[18] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *arXiv preprint arXiv:1506.02640*. Retrieved from https://arxiv.org/pdf/1506.02640

[19] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer M., Heigold, G., Gelly, S., Uskoreit, J., & Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*. Retrieved from https://arxiv.org/pdf/2010.11929

[20] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L & Polosukhin, I. (2017). Attention is all you need. *arXiv preprint arXiv:1706.03762*. Retrieved from https://arxiv.org/pdf/1706.03762

[21] Sonata, I., Heryadi, Y., Wibowo, A., & Budiharto, W. (2023). End-to-end steering angle prediction for autonomous car using vision transformer. *CommIT (Communication and Information Technology) Journal, 17*(2). https://journal.binus.ac.id/index.php/commit/article/view/8425

[22] Chen, S. (2017). *Driving datasets* [Data set]. GitHub. https://github.com/SullyChen/driving-datasets

[23] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., & Koltun, V. (2017). CARLA: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*. Retrieved from https://arxiv.org/pdf/1711.03938