Exploring Autonomous Driving: a Deep Learning Approach

Albert Albesa[1]
[1] *University of Nottingham, School of Physics and Astronomy, Nottingham, United Kinddom*
(Dated: May 28, 2021)

The former is the report for a *Machine Learning in Science II* module project, as part of the *Master of Science in Computational Neuroscience, Cognition and AI* at the *University of Nottingham*.
We study different models in the literature that can be potentially useful to tackle a Machine Learning problem. In particular, the problem is set in the context of self-driving cars, where the input are images taken from a 3D printed circuit and the output are the steering angle and velocity of the car. The reviewed models are NVIDIA, LANENET, YOLOv3 and DETR. Later, we present two particular architectures (SADS and HAPPS) that use, modify and combine the aforementioned models to make predictions on the dataset.
All this was possible thanks to an organized competition that motivated the team and helped it try to find what the different aspects needed polishing during the training process.

Keywords: self-driving cars, autonomous driving, computer vision, lane detection, object detection, CNN

I. INTRODUCTION

The colossal growth of Deep Learning (DL) in the past decade [8] has left very few areas of technology unchanged, and autonomous driving has not been an exception for this. In less than 20 years, we have transitioned from the first DARPA Grand Challenge [14], where all of the participants failed to autonomously cross the Mojave desert in 2004, to cars being allowed to run on *autopilot* in more and more places everyday around the globe.
Systems designed for autonomous driving are usually divided into a perception system and a decision making (DM) unit [1]. The development of Convolutional Neural Networks (CNN's) [6, 9] and subsequent advances in Computer Vision (CV) since ImageNet [5], have raised machines abilities to accurately classify and locate objects in images at *super-human* performance levels and help the car understand what is happening in the outside world together with other sensory information processing (such as odometry). Nevertheless, from a Machine Learning (ML) perspective, the self-driving problem poses a series of challenges that reside in the higher-cognition realm and go beyond object detection. In this last sense, Decision Making (DM) is possibly one of the hardest problems for machines (precisely because we want *their* decisions to be similar to *ours*). It is not by chance, thus, that companies like DeepMind have put Reinforcement Learning (RL, the mathematical foundation for most (DM) models [21]) on the spotlight as the path to achieve Artificial General Intelligence (AGI). The tasks associated to a car DM system can be hierarchically classified into *route planning, behavioural decision making, motion planning* and *vehicle control* [15] and involve taking into account an almost endless list of items that can be as far away from each other as understanding the physics of car motion and predicting human behaviour.
The present study approaches autonomous driving in an end-to-end fashion to train a neural network capable of moving around in a few simplified 3D printed circuits. We first introduce in more detail the nature of the problem, together with examples of data points that illustrate the type of tasks that the model can be asked to do. Later, we describe our proposed model, together with those on top of which it is built. Finally, we discuss the results obtained both from the perspective of a ML problem (in terms of performance metrics) and of a system that has been built to also be tested in the real world.

II. TASK & DASTASET

The dataset contains images taken by a car[1] while driving in a small circuit with 3D printed objects as traffic lights, right and left turn signs, pedestrians or obstacles (figure 1). The training data contains, together with the images, the steering angle (θ) and speed of the car (v) during manual control, with $\theta \in [0, 1]$ and $v \in \{0, 1\}$ ($\theta = 0$ corresponds to the car steering completely to the left and $\theta = 1$ to the car steering completely to the right, while $v = 0$ means the car should stop and $v = 1$ that it should keep moving on). Several teams competed with their DL models in achieving the best performance through a Kaggle competition. In addition, a live testing in which real cars had to complete different challenges was also organized.

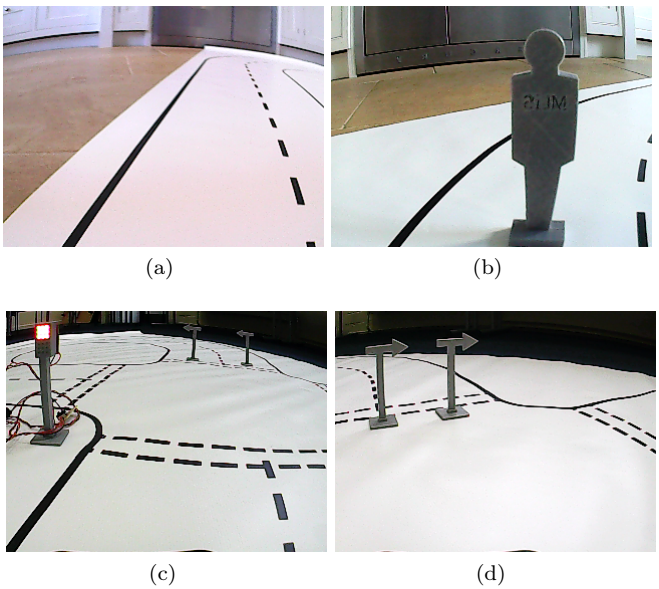

Figure 1. Examples of driving scenes found in the dataset: (a) no specific object on a straight road ($v = 1$, $\theta \approx 0.5$), (b) an obstacle in the road ($v = 0$), (c) red traffic light ($v = 0$), (d) right turn sign ($\theta \approx 1$).

[1] SunFounder PiCar-V kit V2, equipped with a Raspberry Pi and a Coral Edge TPU [7]

III. METHODS

A. Models in the Literature

A.1 NVIDIA

In reviews on the topic of self-driving cars one of the most preponderant appearances is that of the NVIDIA model [2]. They propose an extremely simple architecture, with a 5 layers convolutional backbone and only 3 linear layers to elaborate a decision based on the features extracted in the different channels received by the convolutional layers. They train the network end-to-end on thousands of hours of driving data in which for each image a steering and velocity command is annotated and it performs outstandingly well. It should be noted how a model like this rises as a natural choice for our task due to the extraordinary resemblance between their dataset (images mapped to steering commands) and ours.

A.2.1 LANENET

In [13], the LANENET architecture is presented. This model takes as input a driving scene and outputs N_l sets of points that correspond to the coordinates of the pixels predicted to conform each lane. Moreover, these points are processed by H-NET, which projects them onto a *bird's eye view* and then fits the projected space into a 3^{rd} order polynomial. The architecture of LANENET is depicted in figure 2.

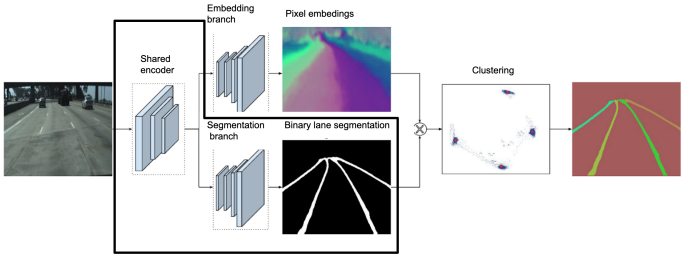

Figure 2. LaneNET architecture

From its different components, due to the complex training pipeline and required labeling, we implemented only the binary segmentation unit (indicated in figure 2 with a black line), which is in turn based on ENET [16].

A.2.2 ENET

ENET uses an encoder-decoder architecture to perform image semantic segmentation. For every training example, two labels must be provided: one is the true label that contains, for every pixel, the class to which it belongs; the other is a blurred, less accurate version of the label. The loss is the weighted sum of categorical cross-entropies, so that the contribution of a pixel with label c_i is multiplied by:

$$w(c_i) = \frac{1}{\ln\left(p(c_i) + c\right)} \tag{1}$$

where $p(c_i)$ is the probability of occurrence of the class across the dataset (number of pixels labelled as c_i divided by the total number of pixels) and c is a constant (originally set to 1.02).

A.3 YOLOv3

Based on [17], YOLOv3 perfected real-time object detection and placed its predecessor on the top of performance and inference time metrics [18].
As many other object-detection models, it does incorporate some geometric prior on the dataset. In the case of YOLOv3 this is done with anchor boxes, a set of 9 (3 per 3 scales) positions and sizes that are found to better encode the likeliness of a box to exist in that particular position and shape. At the same time, the model splits the image into $S \times S$ grids, and for each it predicts B boxes.
YOlov3 loss can be split in three terms:

$$\mathcal{L}_{YOLO} = \mathcal{L}_{classification} + \mathcal{L}_{localization} + \mathcal{L}_{confidence} \tag{2}$$

The first term corresponds to the classification loss:

$$\mathcal{L}_{classification} = \sum_{i=0}^{S^2} \mathbb{1}_i^{obj} \sum_{c \in classes} \left(p_i(c) - \hat{p}_i(c)\right)^2 \tag{3}$$

where $\mathbb{1}_i^{obj}$ has value 1 if cell i is predicted to contain an object and 0 otherwise, $p_i(c)$ and $\hat{p}_i$(c) are (respectively) the conditional (with prior *cell i containing an object*) true and predicted probabilities of the object belonging to class c. The second is the localization loss:

$$\begin{aligned}\mathcal{L}_{localization} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2\right] \\ + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2\right]\end{aligned} \tag{4}$$

where (x, y) $[(w, h)]$ is the true bounding box center position [height and width] and $(\hat{x}, \hat{y})$ $[(\hat{w}, \hat{h})]$ the corresponding prediction.
The final part is the loss of the confidence score for each bounding box predictor, depicted:

$$\begin{aligned}\mathcal{L}_{confidence} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left(C_i - \hat{C}_i\right)^2 \\ + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} \left(C_i - \hat{C}_i\right)^2\end{aligned} \tag{5}$$

where C is the confidence score and $\hat{C}_i$ is IoU of the predicted bounding box over the labeled box. In equations $(5) and (4) \lambda_{coord} = 5$ and λ_{noobj}=0.5 account for the imbalance in objectness across pixels.

A.4 DETR

Many approaches have been proposed to tackle the problem of object detection in ML. This is not surprising, giving that the high-cognition nature of the task leads to solutions that are usually incomplete, with one or another drawback, and depend on priors that are incorporated to the model. In this sense, Facebook Research has recently proposed DETR (DEtection TRansformer, [3]), which implements object detection in a radically different way from that of preexisting models in the literature. The model performs a direct set prediction over N different boxes, so that each box is assigned a class and a center, width and height.

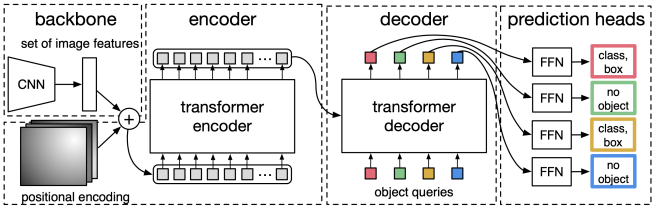

Figure 3. DETR architecture

In figure 3 one can see the main components of DETR: a convolutional backbone that maps an image into a lower-resolution activation map, a transformer encoder-decoder [22] and N independent feed-forward layers, each of which outputs a class and bounding box prediction. The two aspects that make DETR unique are probably the incorporation of attention mechanisms and the proposed loss (which is tightly related to the direct set prediction approach). Given N_O objects labelled in an image, the label set is padded to have N items by adding no-object elements. Then a bi-partite matching (equation (6)) is applied such that the loss between the predicted set and the true set is minimized. Note how predicting more than one box for the same box will inevitable lead to an increase of the loss, as only one of the predicted items will find a counter-partner in the label set. Thus, DETR has to learn itself to perform what in most models is achieved via post-processing with methods as NMS. The permutation of the predictions that contributes to the loss can be formally expressed as:

$$\hat{\sigma} = argmin_{\sigma \in \mathfrak{S}_N} \sum_{i}^{N} \mathcal{L}_{match}(y_i, \hat{y}_{\sigma(i)}) \tag{6}$$

where σ are the different permutations of N elements (set denoted by $\mathfrak{S}_N$), y_i are the labeled tuples of classes and boxes and $\hat{y}_{\sigma(i)}$ are the i^{th} predictions of the model according to σ.
DETR uses a GIoU (Genneralized Intersection over Union) loss that can be expressed as

$$\mathcal{L}_{GIoU}(b_i, \hat{b}_{\sigma\hat{(i)}}) = 1 - \left(\frac{|b_i \bigcap \hat{b}_{\hat{\sigma}(i)}|}{|b_i \bigcup \hat{b}_{\hat{\sigma}(i)}|} - \frac{|B(b_i, \hat{b}_{\hat{\sigma}(i)}) \setminus b_i \bigcap \hat{b}_{\hat{\sigma}(i)}|}{|B(b_i, \hat{b}_{\hat{\sigma}(i)})|} \right) \tag{7}$$

where $b_i \bigcap \hat{b}_{\hat{\sigma}}(i)$ and $b_i \bigcup \hat{b}_{\hat{\sigma}(i)}$ denote (respectively) the intersection and the union of the regions defined by b_i and $\hat{b}_{\hat{\sigma}(i)}$, $|r|$ is the area of r, $B(a, b)$ is the smallest convex hull that encloses a and b and $A \setminus B$ represents the elements of A not contained in B. This loss is identical to classic IoU but adds an extra penalization for boxes that, apart from having 0 intersection, are appart from each other.

B. PROPOSED MODELS

B.1 SADS

SADS stands for *Simple AuDeep Driving System*, as AuDeep was the the name of the fictitious company that our team proposed for the competition. It is built on the convolutional architecture proposed in [2], and a clear inspiration in the modifications and regularization parameters found in [23]. The last includes the additional max pooling and the addition of dropout in the last linear layers, although with some fine tuning regarding our training task.
The linear layers have been specifically designed for our use-case and have 4 common linear layers for extracted features processing, and two additional units specialized independently generating the angle and speed output. The losses that derive from each of the output are also independent, as they correspond to two different ML problems (regression for angle and binary classification for speed). As such, the natural choices for loss are the *Mean Squared Error* (MSE) and *Binary Cross Entropy* (BCE). This particular choice is opposed to the evaluation metrics in the contest, where a global MSE is applied to the output (θ, v).

B.2 HAPPS

Our *Helped AuDeep Piloting Premium System* (HAPPS) is based on the SADS architecture, but instead of being fed directly the training images it takes as input a *helped* version of them.

Table I. Painting Algorithm Dictionary

Channel	Obstacles	Right	Left	Red	Lane
R	100	0	0	255	255
G	0	100	0	0	255
B	0	0	100	0	100

We independently trained the aforementioned literature models, so that for a given image we got a prediction of the different objects present, together with the lane pixels. We designed a *painting algorithm* that takes this information and paints the original image accordingly:

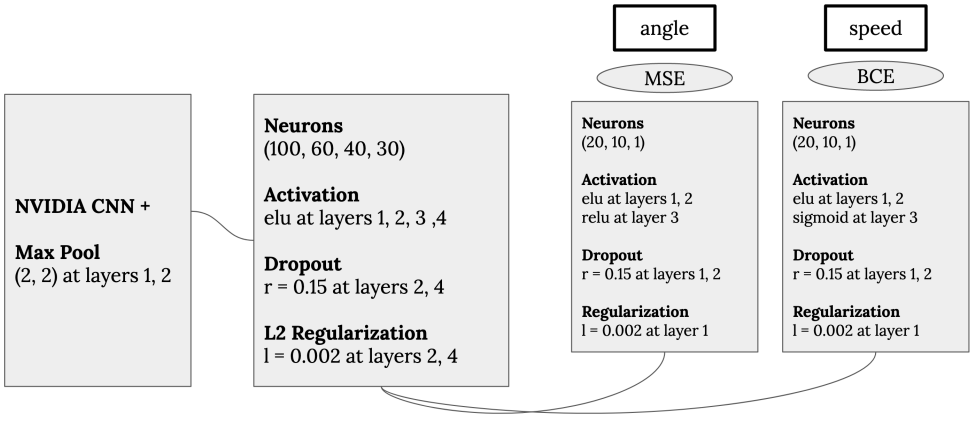

Figure 4. SADS architecture

IV. RESULTS

A. Lane Detection

Final lane detection models are ENET-L (for Lane) and ENET-R (for Road). Both are based on an ENET architecture but were trained on a different task. The output layer changed to a sigmoid and the loss to BCE (to perform binary segmentation). The weight in equation (1) was further fine-tuned and found to be optimal at twice its original value. For labeling, we used the pyhton package LabelMe [19], which extends labeling from traditional bounding boxes to arbitrary polygons.

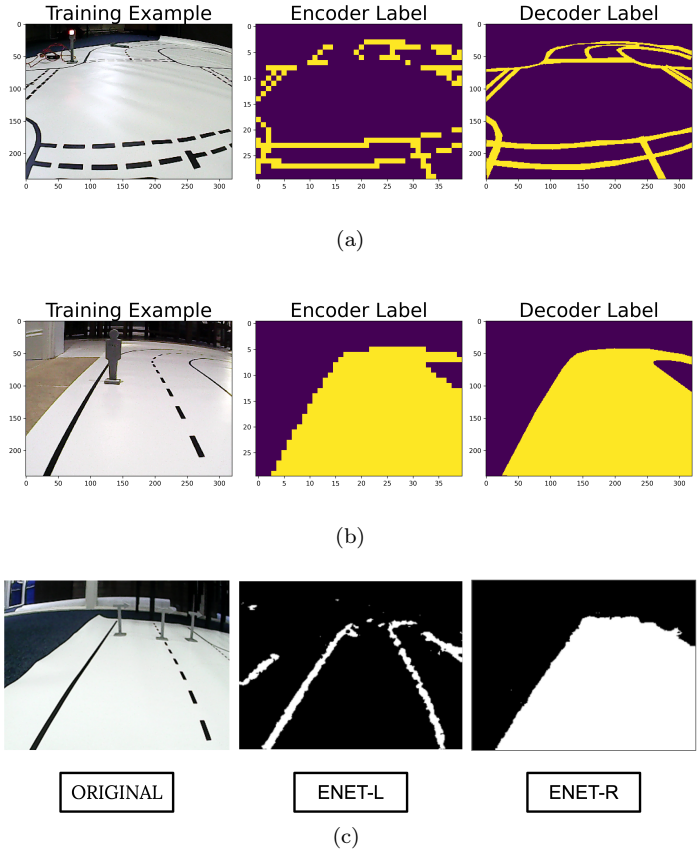

Figure 5. (a) Labels for ENET-L (b) Labels for ENET-R (c) predictions of the models on unseen data

A.1 ENET-L

ENET-L was trained to detect the pixels of the image belonging to the lane (figure 5(a)). In figure 7 one can see the two-stages training process. First the loss and accuracy of the encoder are improved, while finally the two units of the model converge to a good performance state. In figure 5(c) we can see the prediction of the model on an unseen data point. We intentionally selected this image because it shows a predicted non-existing lane (ENET-R specialized in finding high-contrast edges).

A.2 ENET-R

The task of ENET-R was to predict whether the pixels belonged to the inside or outside the road (figure 5(b)). We modified the task to this for two reasons: one, as aforementioned, we wanted to force the algorithm to learn to generalize the concept of road and not only find lane-looking edges. In addition, thinking of its output in the context of the HAPPS model, painting all the road could help the model distinguish when an obstacle is inside or out of it by correlating the mixed colour with a stop. The training curves and performance on unseen data can be seen, (respectively) in figures 7(b) and 5(c).

B. Object Detection

B.1 YOLOv3

We trained YOLOv3 to detect different objects in the dataset: *obstacle, right, left, green* and *red*.

Table II. Performance Metrics of YOLOv3

	Obstacles	Right	Left	Green	Red
TP	814	436	242	84	178
FP	23	0	0	0	0
FN	21	19	24	56	139
Total #	835	455	266	140	317
Precision	0.97	1	1	1	1
Recall	0.97	0.95	0.90	0.6	0.56

We trained the algorithm until convergence (figure) with a final loss slightly higher than the recommended in most guides (0.15 vs 0.06). In figure 6 predictions on unseen images are displayed.

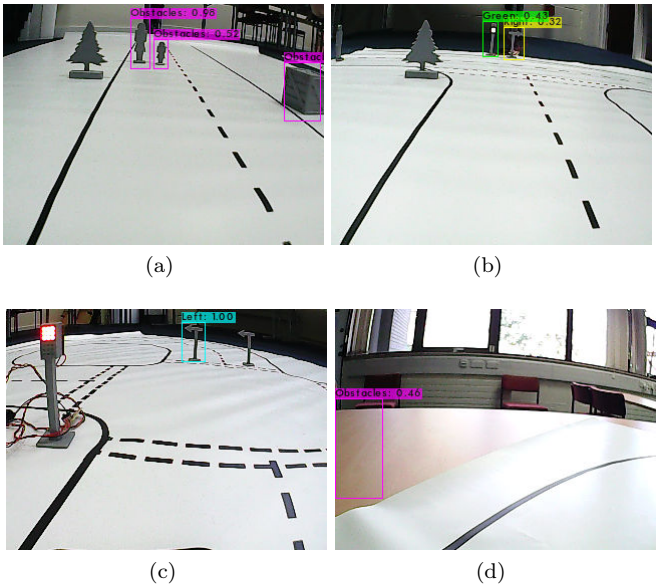

Figure 6. Results from YOLOv3. (a) detection of three obstacles. (b) detection of small images. (c) failed red light detection; (FN) (d) detection of no-object obstacle; (FP)

B.2 DETR

Training of the DETR model did not lead to successful results: in all our experiments the predicted objects were always no-object class.

Due to the novelty of the model it was hard to find ready-to-use models and dataset processings. Only setting up the training environment and generate a compatible dataset took the team more than 3 full days.

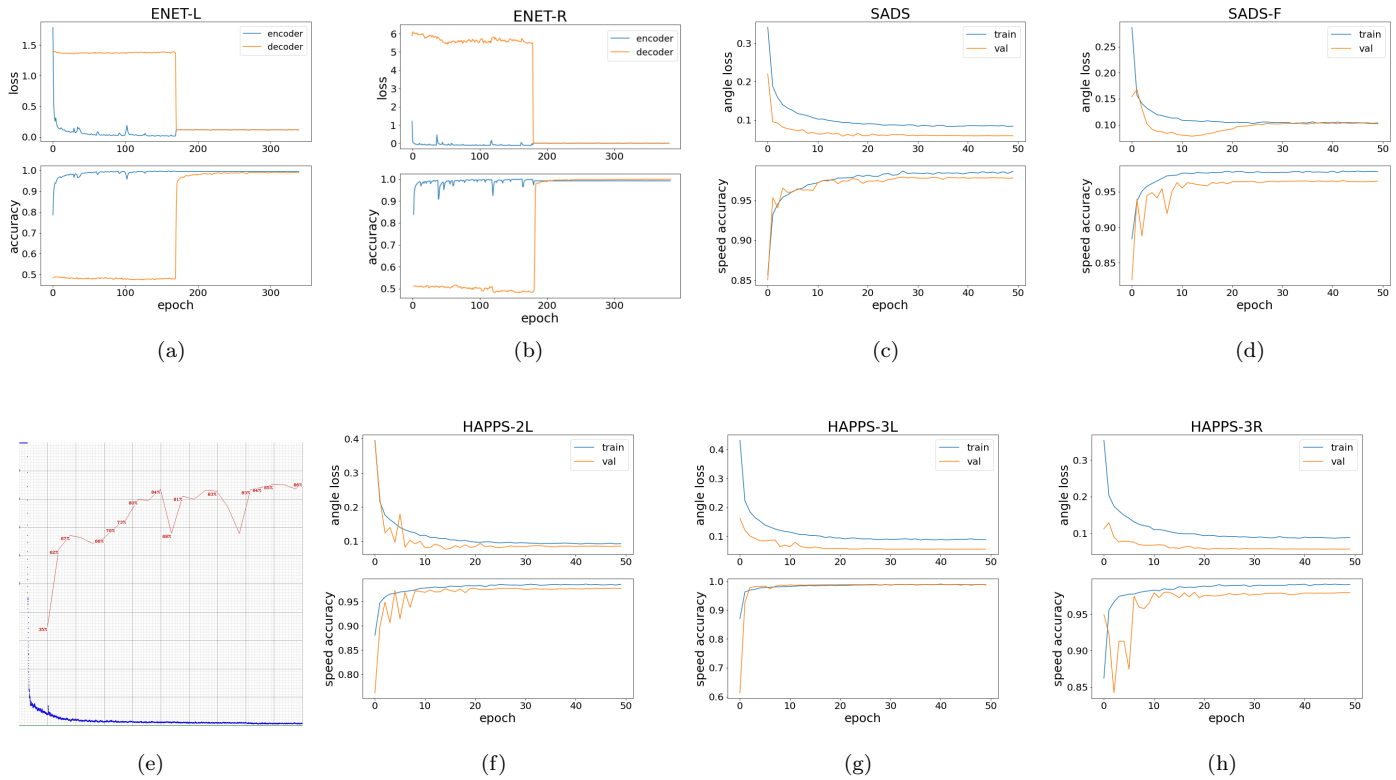

Figure 7. Learning curves for: (a) ENet-L (b) ENet-R (c) SADS (d) SADS-F (e) loss and mAP curve for YOLOv3; blue line depicts the loss curve and the red line denotes the mAP value for every 1000 iterations (f) HAPPS-2L (g) HAPPS-3L (h) HAPPS-3R.The minimum loss and maximum accuracy value for each models are stated in table III.

We tried different hyperparameters and training schemes. A clear candidate to be changed was the number of object-queries, given that our task never involved more than 4 or 5 objects present in an image and N was originally set to 100. We also changed the learning rate and the trainability of the different layers.
One possible explanation is that our dataset was simple insufficient. Pipelines for long-standing object detectors as YOLO have been extremely optimized over years to perform well even when the training data is scarce, which does not apply to DETR (we required about 15 minutes of training per epoch on the example dataset and our training data led to epochs of less than 30 seconds).

C. MLiS2 Self-Driving Competition

C.1.1 SADS

SADS model was trained end-to-end on the original training data and labels. Besides being the first model we explored, it set quite a high standard in terms of performance, as seen in figure 7(c) and table III.

C.1.2 SADS-F

To train SADS-F we doubled the dataset fed to SADS by adding to each image its flipped version. Together with this, the labels were processed such that a flipped image was assigned an angle $\theta_F = abs(1 - \theta)$. In figure 7(d) one can see a dip in the learning curve of the angles. We hypothesise that the rebump is a sign of the model overfitting into an flipped yet task-asymetric dataset.

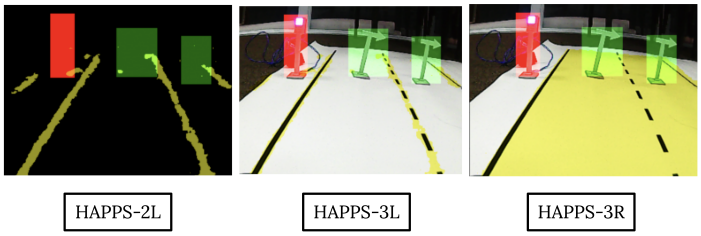

Figure 8. Different versions of HAPPS. HAPPS-2L corresponds to overlay of YOLOv3 colored boxes on ENET-L output. HAPPS-3L is the overlay of HAPPS-2L on the input image. HAPPS-3R differs from HAPPS-3L in that it uses the output of ENET-R.

C.2.1 HAPPS-2L

This version of HAPPS was the first of all. Results (figure 7(f)) show an unstable training, together with lower performance metrics (table III). This is due to the fact that both lane and object detection models were imperfect and the model was not receiving any input besides the output of the two.

C.2.2 HAPPS-3L

HAPPS-3L was the best performing model among all. Its accuracy reached a 0.989% (table III) and its angle loss was comparable to best performing models. Its learning was by far the most stable (figure 7(g)).

C.2.3 HAPPS-3R

Performance for this model did not show an improvement with respect to HAPPS-3L, and the training was more unstable (figure 7(h)). This was probably due to the difficulty of the task assigned to ENET-R, which had to better generalize the concept of road. We speculate that the model would have outperformed all the others with a proper road detection training (more labelled data), as we did not have the time to test this hypothesis.

Table III. Performance Metrics for Competition Models

Model	MSE (θ)	accuracy (v)	GFLOPS/FPS
SADS	0.018	0.979	12/32
SADS-F	0.035	0.965	12/32
HAPPS-2L	0.024	0.960	74/14
HAPPS-3L	0.018	0.989	74/14
HAPPS-3R	0.018	0.980	74/14

C.3.1 - Kaggle Score

The final score obtained in Kaggle was of 0.01827 (Private Leaderboard), which was in accordance with the Public Learderboard score (0.01856). This score granted us the 4^{th} position among 18 different teams.
The submissions corresponding to SADS model did never achieve a score over 0.04. In the end, incorporating object detection through HAPPS allowed us to jump several positions. Nevertheless, had we started submitting and finetuning HAPPS predictions earlier we would have been able to obtain a better score.

C.3.2 - Live Testing

The Live Testing comprised several challenges for which points were awarded (provided successful performance), adding up to a total of 24. Our SADS model achieved a score of 21, which granted it a (shared) 2^{nd} position.
Given that we submitted the SADS model with the intention to have a simple model with a reduced inference time, it was not equipped with specific object detection mechanisms. It is not surprising that the tests that it failed were highly related to object detection: stopping at a red traffic light and doing a left turn (this was done but poorly, too late in time).

V. DISCUSSION

In this report we have reviewed some of the traditional and also state-of-the-art models that can be used in object detection and final decision making in self-driving cars. Moreover, we have also proposed different architectures based on them that have been specifically designed to perform on an original task.
The nature of the dataset and the task has put aside the possibility of considering other models and techniques. With a dataset that includes temporal information (as video recordings, for example) Recurrent Neural Networks [10] could have been utilized to make decisions based on both real-time inputs and past information [4]. At the same time, for tasks that include asking the car to drive from one particular point to another, Deep Reinforcement Learning [12] could have been implemented [20].
Our results show how specific architectures for object detection can help autonomous driving models and make them perform better. A possible line of future research would be more systematically testing the magnitude of the effect of each of these systems independently (for example: how does performance exactly improve for a particular lane detection model, and with what confidence?).
At the same time, it should also be noted how adding this models adds a lot of computational complexity. It seems reasonable that this will also be applicable to real-life models, where performance has such little improvement margin that every time the ratio of software and hardware enhancements have a very low impact in terms of their cost. Furthermore, we have seen also the different challenges that model integration can pose. When we wanted to add object detection to our model we realised it was not trivial to choose what should be the information flow. For example, how would the model do if the bare output (class and position and shape of the bounding box) was just concatenated to the final activation map outputted by the convolutional layer in SADS? Geometrical information, which in most of the vector would be encoded in its position, would be mixed with regression-like (output of object detection). Would the linear layers learn to process both types of input together? Another clear possibility for further research would be investigating this issue, together with the importance of semantic orthogonality/overlapping in the color dictionary presented in table I.
Another question that rises naturally from trying to understand (from a human perspective) what are the strengths and weaknesses of the model is its explainability, which will possibly play a very important role in the roadmap for autonomous driving to become a reality [24]. An interesting concept paired with this is the known as the long-tail problem [11], which describes how no-matter the accuracy of the model there will always exist very unlikely situations that for which it will not respond adequately. Ironically, this makes explicit the double standards that apply for considering a self-driving car to be commercial, from which is expected much from than from a human being.
Finally, I would like to add a note regarding the academic nature of this project. As such, its main goal has not been the development of a model or training method *per se* but educational instead. In this sense, the results could have not been better, as the learning curve has been far from flattened at all points of the process. If any, the only drawback has been not having more time to delve deeper into this first contact with the world of Deep Learning.

[1] Badue, C., Guidolini, R., Carneiro, R. V., Azevedo, P., Cardoso, V. B., Forechi, A., Jesus, L., Berriel, R., Paixao, T. M., Mutz, F., et al. Self-driving cars: A survey. *Expert Systems with Applications* (2020), 113816.
[2] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).
[3] Carion, N., Massa, F., Synnaeve, G., Usunier, N.,

Kirillov, A., and Zagoruyko, S. End-to-end object detection with transformers. In *European Conference on Computer Vision* (2020), Springer, pp. 213–229.

[4] Chen, S., Zhang, S., Shang, J., Chen, B., and Zheng, N. Brain-inspired cognitive model with attention for self-driving cars. *IEEE Transactions on Cognitive and Developmental Systems 11*, 1 (2017), 13–25.

[5] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (2009), Ieee, pp. 248–255.

[6] Fukushima, K., and Miyake, S. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.

[7] Hawks, B., Jasal, P., Wang, M., and Nord, B. Real-time machine learning inferencing with edge computing devices from google and intel. Tech. rep., Fermi National Accelerator Lab.(FNAL), Batavia, IL (United States), 2019.

[8] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature 521*, 7553 (2015), 436–444.

[9] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural computation 1*, 4 (1989), 541–551.

[10] Lipton, Z. C., Berkowitz, J., and Elkan, C. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019* (2015).

[11] Makansi, O., Cicek, Ö., Marrakchi, Y., and Brox, T. On exposing the challenging long tail in future prediction of traffic actors. *arXiv preprint arXiv:2103.12474* (2021).

[12] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[13] Neven, D., De Brabandere, B., Georgoulis, S., Proesmans, M., and Van Gool, L. Towards end-to-end lane detection: an instance segmentation approach. In *2018 IEEE intelligent vehicles symposium (IV)* (2018), IEEE, pp. 286–291.

[14] Ozguner, U., Stiller, C., and Redmill, K. Systems for safety and autonomous behavior in cars: The darpa grand challenge experience. *Proceedings of the IEEE 95*, 2 (2007), 397–412.

[15] Paden, B., Čáp, M., Yong, S. Z., Yershov, D., and Frazzoli, E. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles 1*, 1 (2016), 33–55.

[16] Paszke, A., Chaurasia, A., Kim, S., and Culurciello, E. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147* (2016).

[17] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 779–788.

[18] Redmon, J., and Farhadi, A. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767* (2018).

[19] Russell, B. C., Torralba, A., Murphy, K. P., and Freeman, W. T. Labelme: a database and web-based tool for image annotation. *International journal of computer vision 77*, 1-3 (2008), 157–173.

[20] Shalev-Shwartz, S., Shammah, S., and Shashua, A. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295* (2016).

[21] Sutton, R. S., and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

[22] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *arXiv preprint arXiv:1706.03762* (2017).

[23] Wang, Y., Liu, D., Jeon, H., Chu, Z., and Matson, E. T. End-to-end learning approach for autonomous driving: A convolutional neural network model. In *ICAART (2)* (2019), pp. 833–839.

[24] Zablocki, É., Ben-Younes, H., Pérez, P., and Cord, M. Explainability of vision-based autonomous driving systems: Review and challenges. *arXiv preprint arXiv:2101.05307* (2021).