From Pixels to Actions:
Learning to Drive a Car with Deep Neural Networks

Jonas Heylen* Seppe Iven‡ Bert De Brabandere‡
Jose Oramas M.‡ Luc Van Gool‡ Tinne Tuytelaars‡
*TRACE-Leuven ‡KU Leuven, ESAT-PSI, imec

Abstract

The promise of self-driving cars promotes several advantages, e.g. they have the ability to outperform human drivers while being safer. Here we take a deeper look into some aspects from algorithms aimed at making this promise a reality. More specifically, we analyze an end-to-end neural network to predict a car's steering actions on a highway based on images taken from a single car-mounted camera. We focus our analysis on several aspects which could have a significant impact on the performance of the system. These aspects are: the input data format, the temporal dependencies between consecutive inputs, and the origin of the data. We show that, for the task at hand, regression networks outperform their classifier counterparts. In addition, there seems to be a small difference between networks that use coloured images and ones that use grayscale images as input. For the second aspect, by feeding the network three concatenated images, we get a significant decrease of 30% in mean squared error. For the third aspect, by using simulation data we are able to train networks that have a performance comparable to networks trained on real-life datasets. We also qualitatively demonstrate that the standard metrics that are used to evaluate networks do not necessarily accurately reflect a system's driving behaviour. We show that a promising confusion matrix may result in poor driving behaviour while a very ill-looking confusion matrix may result in good driving behaviour.

1. Introduction

Neural networks have gained a lot of popularity from their successes in the ImageNet Large Scale Visual Recognition Competition, e.g.[15]. They have since been applied in many different areas, often resulting in substantial improvements. In this paper, we develop an end-to-end system to control a self-driving car. Given the wide range of systems and components required for self-driving vehicles, we focus on a simplified version of the problem focusing only

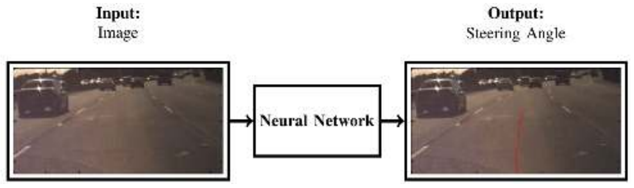

Figure 1: Process flow of our system.

on the steering of the car while it is driving on a highway. This is illustrated in Figure 1.

Most state-of-the-art works follow a mediated perception approach, which is based on object detection to make driving decisions [3, 17, 19]. In contrast, a reduced amount of research addresses direct end-to-end mapping of images to driving actions. Therefore, the main contribution of this work is an analysis on several important aspects that must be taken into account when developing end-to-end vision systems for autonomous navigation.

The first aspect we analyze is the format of the input data that is being fed into the network. We look into the influence of the quantization granularity of the steering wheel angle's measurements onto the system's performance. We also verify the color format of the input images and compare performance when using colour VS. grayscale images. This shows us to what extent the system can make use of the information that lies within the colour of the images.

The second aspect analyzes the effect of exploiting temporal consistency that can be found in successive input images. We compare two techniques that can give the network more capabilities to utilize this information. The first technique consists of concatenating multiple subsequent images and feeding them to the network as a single input. This leads to an increased input size, but the architecture of the network remains the same. The second technique consists of inserting recurrent neural network layers into the architectures that we use. By definition, recurrent layers can retain information between consecutive inputs and thus utilize the temporal information.

The final aspect is the origin/nature of the data. We experiment with the applications of artificially generated or simulated data, which can hold many advantages over real-world data. If a simulator is sufficiently advanced and mimics the circumstances of the real world well enough, it can be used to train the neural networks and this would reduce the need for real-world data to train on. In a simulator it is very simple and cheap to gather data and the settings of this data are easy to change. This leads to bigger and more diverse datasets, which can be used to train more robust and better performing networks.

Finally, we revisit the question of whether the performance metrics that are used on datasets, are a good indicator of a network's real driving behaviour. This can only be reviewed by actually driving on the road or in a simulator.

The remaining sections of this paper are organized as follows: Section 2 positions our paper with respect to related work. Section 3 introduces the methodology followed in our analysis. Section 4 presents the experiments conducted as part of our analysis and their results. Finally, Section 5 draws conclusions.

2. Related Work

The related work regarding self-driving cars can be divided into two categories: mediated perception approaches and end-to-end approaches. We position our work with respect to these two categories.

Mediated Perception Approaches: Most state-of-the-art systems use a mediated perception approach [3, 17, 19]. These approaches rely on the detection and classification of surrounding objects such as traffic signs, cars, roads, buildings and pedestrians. They parse the entire scene into an internal map of the surroundings and use this to make driving decisions. A common practice in this type of approaches is that objects and other scene elements that are considered "relevant" need to be pre-defined. Towards this goal a significant amount of efforts have focused on designing specialized algorithms to detect individual objects [2, 5, 6, 9, 12, 18, 29, 33]. Another common characteristic of mediated perception approaches is their requirement of multiple sensors for reliable object detection and classification. Examples of these sensors include cameras, lasers and radar. Mediated approaches usually consist of two steps: i) detection of "relevant" visual elements, and ii) decision making based on those elements. This has both advantages and disadvantages. One the one hand, a disadvantage is that it is possible that a designer fails to identify certain relevant objects. Moreover, it is also possible that useful information may get lost between the two steps. The objects that are to be detected are usually hand-picked. Because of this, certain detected objects may not be important for the decision making while other meaningful objects may not be detected and this may deteriorate the system's performance. On the other hand, an advantage is that this level of indirection makes it easier for the network to focus on certain details. For example, it is important to detect if a pedestrian has the intention to cross the street and a mediated perception approach can use a network that specializes in detecting this. But for an end-to-end network, it may be difficult to learn that it is important or to notice this type of situations. Different from mediated perception approaches, we focus on end-to-end systems where the network is given the task of identifying which visual elements are important for the task at hand. Therefore, no object or any other visual element needs to be pre-defined or hand-picked. In addition, we focus our analysis on the setting where the only input sensor is a colour camera.

End-to-end Approaches: In end-to-end systems, images are directly mapped to driving decisions with the use of machine learning algorithms. Our system uses this approach. Examples of such approaches are [1, 14, 16, 21, 22]. Sometimes multiple cameras are used to create recovery cases or a simple real-world simulator. As said earlier, a disadvantage of end-to-end approaches is that they lack a second processing step or controller that makes decisions. Because of this, the system does not keep track of the bigger picture. This makes it difficult to teach the network certain specific things, for example to abruptly avoid any children that run in front of the car. This is difficult because unless the dataset is artificially created, there are few such situations in the dataset while many different situations should be present to create a robust system. Another disadvantage is that the driving behaviour in the training data has a direct influence as the network learns this imperfect driving style, such as driving too close to the right lane markings. If possible, the images and measurements should be very carefully selected or corrected. This is less of a problem for mediated perception approaches than it is for end-to-end approaches.

Other Related Problems: An additional motivation for the use of end-to-end systems can be found in related problems. [13] uses an end-to-end neural network based on camera images to fly a drone through a room. They stress that using pretrained networks is a good alternative to learning from scratch for end-to-end networks. It saves training time and requires less data because it is less prone to overfitting. They also stress that training LSTM networks using a limited time window produces better performance than when training it on all previous input samples. Moreover, [13] indicates that there is a clear trend in which LSTM networks outperform standard feedforward networks. [34] also used LSTMs combined with other techniques to tackle end-to-end driving. Another observation is that recovery cases have a big impact on performance. Following the same directions, [25] demonstrates that it is possible for networks trained on simulation data to be generalized to the real world. Taking these works into account, it is plausible to assume that many observations and conclusions drawn from the autonomous drone problem, such as the generalization of simulators and the better performance of LSTM networks, also hold for our problem. Finally, motivation for incorporating inter-frame dependencies is found in language modeling. Just like different words in a sentence are also related in order to convey meaning, the input images are temporally dependent because they are consecutive frames of a video. It is proven that LSTMs can outperform standard feedforward neural networks on such tasks [30, 31].

3. Methodology

Our basic system is set-up as follows: images from the camera are fed into the network and the network predicts the steering angle from these image(s). During training, the steering wheel angle measurements, i.e. annotations, are also fed to the network. An illustration of our system is given in Figure 1.

Network Architectures: Throughout our experiments, the neural networks that we use can vary in two areas: their main architecture and their output layer. The main architecture is a variation of either the NVIDIA [1], AlexNet [15] or VGG19 architecture [28]. Note that for the Alexnet architecture, we removed the dropout of the final two dense layers and reduced their sizes to 500 and 200 neurons as this resulted in better performance. The output layer of the network depends on its type (regression or classification) and, for a classification network, on the amount of classes. In our analysis we conduct experiments with both, classification and regression, types. For the case of the classification type, we quantize the steering angle measurements into discrete values, which represents the class labels. This quantization is needed as input when training a classifier network and allows to balance the data through sample weighting. This weighting acts as a coefficient for the network's learning rate for each sample. A sample's weight is directly related to the class that it belongs to when quantized. These class weights are defined as 1 divided by the amount of samples in the training set that belong to that class, multiplied by a constant so that the smallest class weight is equal to 1. Sample weighting is done for both classifier networks and regression networks. Note that for the latter, the class is used, to which the continuous value would be mapped. This weighting is done to ensure that the network is equally trained on all classes, in the hope that it learns to handle all these different situations well. Otherwise, the network might be biased toward a certain class.

Dataset: We train and evaluate different networks on the Comma.ai dataset [26], which consists of 7.25 hours of driving, most of which is done on highways and during daytime. Images are captured at 20 Hz which results in approximately 552,000 images. We discarded the few sequences that were made during the night due to their high imbalance when compared to those captured during daytime. In addition, in order to focus on sequences with continuous / uninterrupted driving, we limit ourselves to only considering images that were captured while driving on highways. The remaining data is then split into two mutually exclusive partitions: a training set of 161,500 images and a validation set of 10,700 images. This is done on a random per-file basis to ensure independence between training and validation and to ensure that both sets contain various traffic and weather conditions. These two datasets are used in all conducted experiments.

Performance Metric: We evaluate performance of our networks using the following performance metrics: accuracy, mean class accuracy (MCA), mean absolute error (MAE) and mean squared error (MSE) metrics. We base our conclusions on the MSE metric, since it allows us to take the magnitude of the error into account and assign a higher loss to larger errors than MAE does. This is desirable since this may lead to better driving behaviour, as we assume that it is easier for the system to recover from many small mistakes than from a few big mistakes. A large prediction error could result in a big sudden change of the steering wheel angle. For example, larger errors create dangerous situations as the car might swerve onto an adjacent lane or go off-road.
For every metric individually, the best performance over all epochs is chosen. These values are then compared between networks and the best network is selected based on the MSE metric. Note that the absolute performances are of relatively low importance to us and that we are more interested in the relative performances between the different network variants in our experiments. After analyzing the high-level aspects of our problem, better performance can later be achieved by optimizing around the results of these experiments.

Models used in this analysis are publicly available at our project website[1].

4. Experiments

In this section, we evaluate the three high-level aspects mentioned earlier. These aspects are the format of the input data (Section 4.1), the temporal dependencies between consecutive inputs (Section 4.2) and the origin of the data (Section 4.3).

4.1. Input Data Format

Quantization granularity: In this first experiment, we look into the influence that the specifications of the class quantization procedure have on the system's performance. These specifications consist of the amount of classes and

[1]`http://homes.esat.kuleuven.be/~jheylen/FromPixelsToActions/FPTA.html`

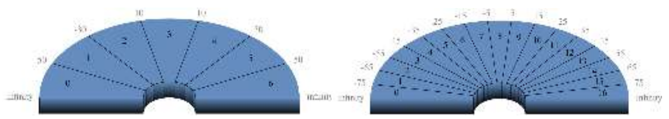

Figure 2: Mapping of angle measurements from continuous values (outside) to discrete class-labels (inside) for 7 and 17 classes, respectively.

the mapping from the input range to these classes. We compare classifier networks with varying degrees of input measurement granularity. We also compare them to regression networks, which can be seen as having infinitely many classes, although using a different loss function. It is plausible that the granularity has a significant impact on the system's performance. For metrics such as global accuracy and mean class accuracy, this is obvious since it is more difficult to choose the right class for a fine quantization configuration that has a higher number of classes. Coarse-grained classes however have a bigger quantization error and this influences the magnitude of the error. For metrics where the magnitude of the error is taken into account, such as MSE and MAE, it is possible that configurations with many fine-grained classes will perform better. Because classifier networks are trained with categorical cross entropy as loss function, we expect them to perform well when compared using class accuracy as metric. On the other hand, regression networks will probably outperform the classifier networks on metrics that take the error distance into account, as their loss function (e.g. MSE) also uses the error distance.

We conduct this experiment by comparing a coarse-grained quantization scheme with 7 classes and a finer-grained scheme with 17 classes. We do this for both classifier and regression networks. The mapping from angles to classes can be found in Figure 2 for 7 and 17 classes, respectively. All of these networks are tested on the three architectures previously explained and evaluated following the methodology discussed in Section 3. The difference between 7 and 17 for regression is in the class weighting. Each sample is given a weight based on their relative occurrences in 7 or 17 classes. (Similar to class weighting for the classification networks.) Also, to be able to compare regression vs classification, the predicted regression outputs were discretized into 7 and 17 classes to calculate MCA in the same way this happened for the classification networks.

The results of this experiment are found in Figures 3 through 6. Several observations can be made. First, it is logica that the coarse-grained scheme scores better on the accuracy and MCA metric. More importantly, we see that regression networks significantly outperform classifier networks on the MAE and MSE metrics, which we have discussed and concluded to be the most important metrics. This aligns with our expectations, since regression networks have a loss function that takes the error magnitude into account. Finally, we notice that class weighting does not have a significant impact on the performance of regression networks. A possible explanation is that this is due to their loss function, which also takes the error magnitude into account. Samples which are less common generally will get a higher loss, as their steering angle is mostly predicted a lot worse than common samples.

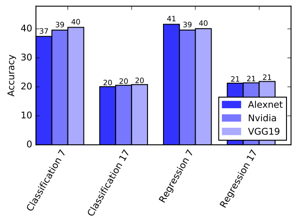

Figure 3: Classification accuracy of the granularity experiment.

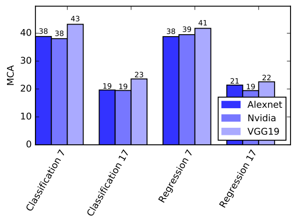

Figure 4: Mean class accuracy (MCA) of the granularity experiment.

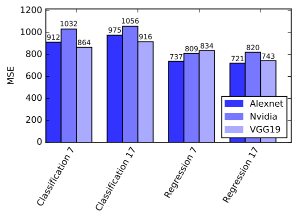

Figure 5: Mean squared error (MSE) of the granularity experiment.

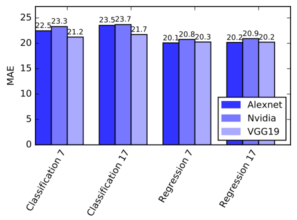

Figure 6: Mean absolute error (MAE) of the granularity experiment.

Image Colour Scheme: Next, we investigate to what extent our system can exploit the colour information that is present in the input images. We start off by comparing coloured images to grayscale ones. The images from the dataset already have the RGB colour scheme. Since our previous network proved regression networks to outperform classifier networks in the problem at hand, we focus this experiment on regression networks.

The results from this experiment can be found in Figures 7 through 10. From these results, it can be observed that there is no significant difference in performance between networks that use coloured and grayscale images as input. This suggests that, for the task at hand, the system is not able to take much advantage of the colour information. Therefore, it is not worthwhile to investigate this aspect any further and compare different colour schemes.

4.2. Incorporating Temporal Dependencies

In this second high-level aspect, we evaluate methods that enable our system to take advantage of information that co-occurs in consecutive inputs. This could lead to signif-

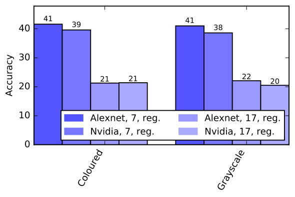
Figure 7: Classification accuracy of the colour experiment.

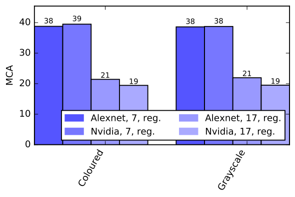
Figure 8: Mean class accuracy (MCA) of the colour experiment.

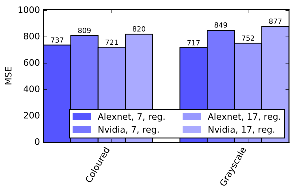
Figure 9: Mean square error (MSE) of the colour experiment.

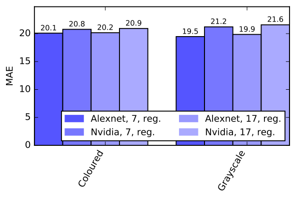
Figure 10: Mean absolute error (MAE) of the colour experiment.

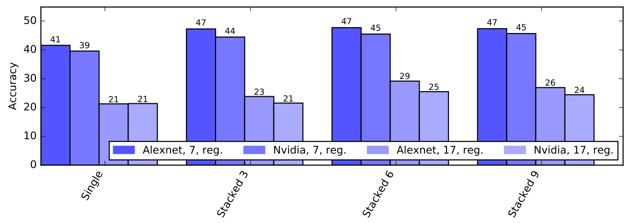
Figure 11: Classification accuracy of the stacking experiment.

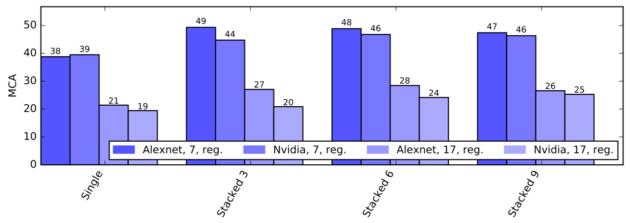
Figure 12: Mean class accuracy (MCA) of the stacking experiment.

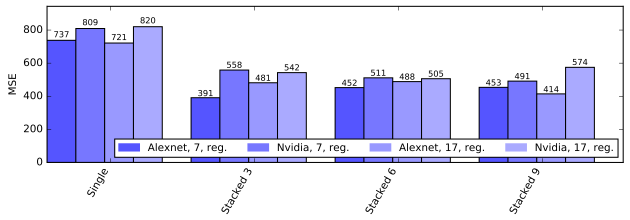
Figure 13: Mean square error (MSE) of the stacking experiment.

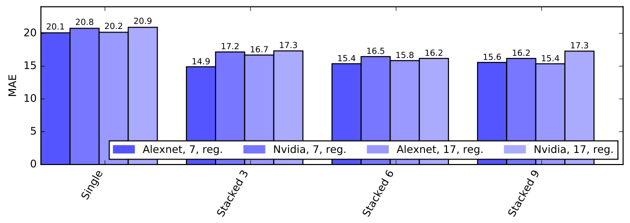
Figure 14: Mean absolute error (MAE) of the stacking experiment.

icant increase in performance as the input images are obtained from successive frames of a video which introduces temporal consistencies.

Stacked frames: In the first method we evaluate, we concatenate multiple subsequent input images to create a stacked image. Then, we feed this stacked image to the network as a single input. We refer to this method as *stacked*. This means that for image i_t at time/frame t , images $i_{t-1}, i_{t-2}, \dots$ will be concatenated. To measure the influence of this stacked input, the input size must be the only variable. For this reason, the images are concatenated in the depth (channel) dimension and not in a new, 4th dimension. For example, stacking two previous images to the current RGB image of 160x320x3 pixels would change its size to 160x320x9 pixels. By doing this, the architecture stays the same since the first layer remains a 2D convolution layer. The only change in our basic pipeline, are the dimensions of the input image and the size of the convolution kernels, whose depth is automatically adjusted as they take the whole depth of the input into account. We expect that by taking advantage of the temporal information between consecutive inputs, the network should be able to outperform networks that perform independent predictions by taking single images as inputs.

For our experiment, we compare single images to stacked frames of size 3, 6 and 9. This means that 2, 5 or 8 preceding images have been concatenated, respectively. The results can be found in figures 11 through 14.

These results show that feeding the network stacked frames increases the performance on all metrics. Looking at MSE, we see a significant decrease of about 30% when comparing single images to stacked frames of 3 images. We assume that this is because the network can make a prediction based on the average information of multiple images. For a single image, the predicted value may be too high or too low. For concatenated images, the combined information could cancel each other out, giving a better 'averaged' prediction.

We see that further increasing the amount of concatenated images only leads to small improvements with diminishing returns. Assuming that the network averages the images in some way, we do not want to increase this amount because the network loses responsiveness. For instance, if a sudden event were to happen, such as a person jumping in front of the car, it would get filtered out by the averaging. The system would only respond to it after several frames, when the event is present in many previous input frames. This would be a reason to desire an increased frame rate. Even though the consecutive images would be more correlated, it would

result in a quicker response of the system.
Based on these observations, in our setting the configuration with 3 concatenated frames is preferable. It offers a significant boost in performance while the system remains relatively responsive.

Recurrent layers: In the second technique, we modify our architecture to include recurrent neural network layers. The type of layers that we use, are Long-term short-memory (LSTM) layers. By definition, these layers allow to capture the temporal information between consecutive inputs. Traditional recurrent NN layers suffer from the vanishing gradient problem [10] but LSTM layers deal with this through the use an internal forget gate. The activation function of the recurrent part of a LSTM block is the unity function. This way the gradient does not vanish nor explode. The LSTM is thus able to retain information over a long period of time. Many different types and modified versions of LSTMs can be found in the literature [7, 8, 11]. We use the version implemented in Keras [4].
The networks are trained on an input vector that consists of the input image and a number of preceding images, just like the stacked frames. Together with our training methodology, this results in a time window. Due to the randomization in our training, this is not a sliding window but a window at a random point of time for every input sample. As explained in [13], this has the effect of decorrelating the samples and leads to higher variance and slower convergence during training, but gets rid of the sequential bias.

We compared many variations of the NVIDIA architecture [1]. We experimented with a configuration where we changed one or both of the two dense layers to LSTM layers, one where we added an LSTM layer after the dense layers and one where we changed the output layer to LSTM. Training these networks from scratch led to very poor performance. Perhaps, this might be caused by the fact that as the LSTM offers increased capabilities, it also has more parameters that need to be learned. We hypothesize that our dataset is too small to do this, especially without data augmentation.
Therefore, we load a pretrained network when we create a LSTM network. This pretrained network is the NVIDIA network variant from our granularity experiment with the corresponding output type. Depending on the exact architecture of the LSTM network, the weights of corresponding layers are copied. Weights of non-corresponding layers are initialized as usual. The weights of the convolutional layers are frozen as they have already been trained to detect the important features and this reduces the training time.
Again we tested the variations of the NVIDIA architecture described above. We found that on pretrained networks, the performance usually remains the same. The results show that the incorporation of LSTM layers did not increase nor reduce the network's performance. It is unclear why this happens and future research could conduct more detailed experiments regarding the incorporation of LSTM layers.

4.3. Application of Simulated Data

A last aspect we investigate is the origin of the data. Up until now, training and evaluation of our system was done using real-world datasets. Here we look into the advantages of a simulator over a real-world dataset and the uses of such a simulator. We research the impact of recovery cases on a network's performance and verify if the performance metrics that are typically used are a good indicator of a network's real driving behaviour.

A simulator brings many advantages. Some examples are that data gathering is easy, cheap and can be automated. Recovery cases can easily be included in the dataset. Infrequently occurring situations can be simulated and added to the dataset. Driving conditions such as the weather and traffic can be set as desired. Testing in simulators is safe, cheap and easy.

Udacity Simulator: First the Udacity simulator [32] is used to generate three datasets. This simulator is very minimalistic and has no other cars, pedestrians, or complex traffic situations. Only simple test-tracks are implemented. The first dataset is gathered by manually driving around the first test-track in the simulator. The second dataset consists of recovery cases only. It is gathered by diverging from the road, after which the recovery to the middle of the road is recorded. This process is repeated many times to get a sufficiently large dataset. A third validation dataset is gathered by driving around the track in the same way as with the first dataset. For the following experiments, the NVIDIA architecture [1] with a regression output is used and no sample weighting is applied during training.

Training on Simulated Data: The first experiment tests the performance of a network trained solely on the first dataset. After training, the best epoch is selected based on MCA. The confusion matrix and MCA are shown in Figure 16. The metrics are comparable to other runs on the real dataset. As the confusion matrix has a dense diagonal, good real-time driving performance is expected. When driving in the simulator, the network starts off quite well and stays nicely in the middle of the road. When it encounters a more difficult sharp turn, the network slightly miss-predicts some frames. The car deviates from the middle of the road and is not able to recover from its miss-predictions, eventually going completely off-track. We conclude that despite promising performance on the traditional metrics, the system fails to keep the car on the road.

Recovery Cases: The second experiment evaluates the influence of adding recovery data. First a new network is trained solely on the recovery dataset. The confusion matrix

Sequence: Network trained from recovery cases only

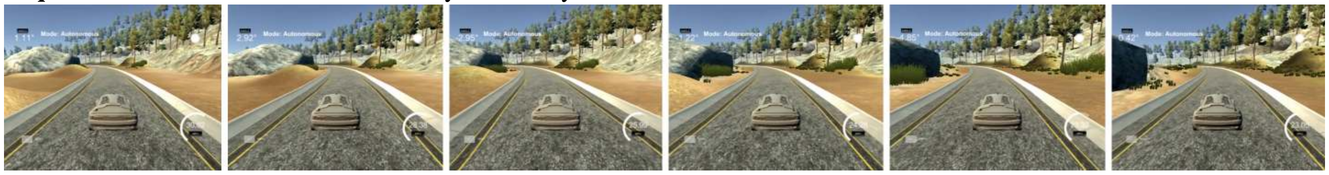

Figure 15: Some video sequences showing the evaluated model driving in the Udacity simulator when considering only data depicting recovery cases. The car does not stay exactly in the middle of the road. The car wobbles softly from one side to the other side of the road during the straight parts of the track. Surprisingly, it handles the sharp turns quite well. Please refer to the supplementary material for the video version of similar sequences.

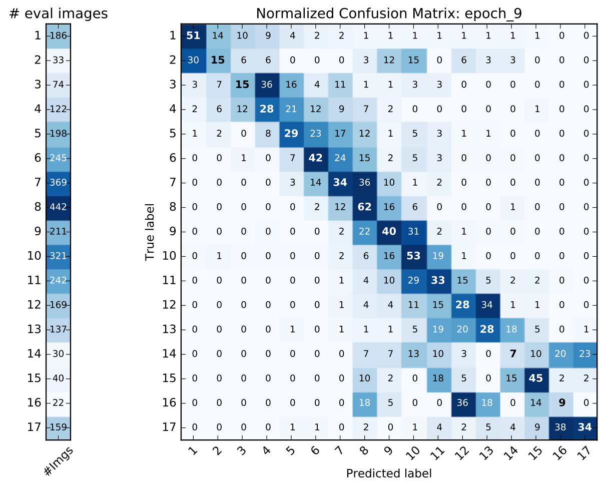

Figure 16: Confusion matrix for NN trained without recovery data. Mean class accuracy (MCA) is 32.5%.

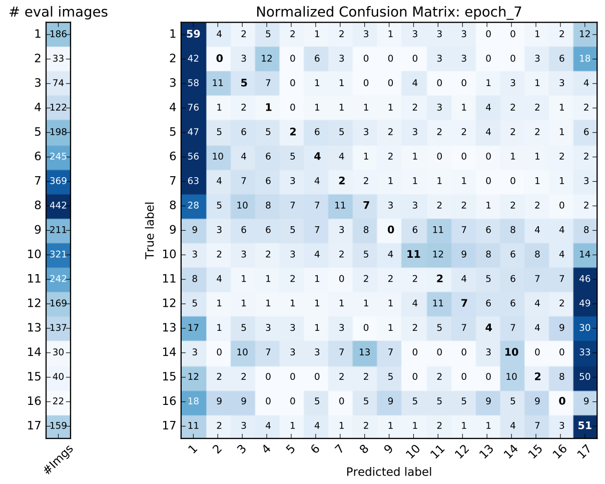

Figure 17: Confusion matrix for NN trained with only recovery data. Mean class accuracy (MCA) is 9.9%.

is shown in Figure 17 together with its MCA. As can be expected, the confusion matrix is focused on steering sharply to the left or right. As it does not look very promising and the MCA is very low, it is expected this network will not perform very well during real-time driving. Despite the low performance on these metrics, the network manages to keep the car on track. The car however does not stay exactly in the middle of the road. Instead, it diverts from the centre of the road, after which it recovers back towards the middle. It then diverts towards the other side and back to the middle again, and so on. The car thus wobbles softly during the straight parts of the track, but handles the sharp turns surprisingly well.

A third network is trained on both datasets and has a confusion matrix similar to the first network. In the simulator, it performs quite well, driving smoothly in the middle of the lane on the straight parts as well as in sharp turns. We conclude that recovery cases have a significant impact on the system's driving behaviour. By adding these recovery cases, the driving performance of the system is improved while its performance on metrics deteriorates. This again suggests that the standard metrics might not be a good tool to accurately assess a network's driving behaviour. Please refer to the supplementary material for videos depicting the cases discussed above.

GTA V Simulator: As an extension to the simplistic Udacity, GTA V [24, 20] is integrated as a more realistic simulator platform. Next to being nearly photo-realistic, GTA V provides a big driving playground of a vast 126 km^2 with various lighting and weather conditions, many different cars, and divers traffic scenes. A big dataset with 42 hours of driving is available at [23]. This data also includes recovery cases. This dataset is composed by 600k images split into 430k training images and 58k validation images. A NVIDIA [1] and an AlexNet [15] regression network, as described above, are trained on the dataset with sample weights based on 17 classes. Since the NVIDIA network performs better, only this one is discussed below. The network shows performance metrics similar to the NVIDIA regression network trained on the real-world dataset. We evaluate real-time driving performance on an easy, non-urban road with clear lane markings. The network performs quite well and stays around the centre of the lane. When approaching a road with vague lane markings, such as a small bridge, the car deviates towards the opposite lane (Figure 18 middle). When it reaches a three-way crossing (Figure 18 bottom), the network can not decide whether to go left or right, as it was equally trained on both cases. Because of this, it drives straight and goes off-road. In an ur-

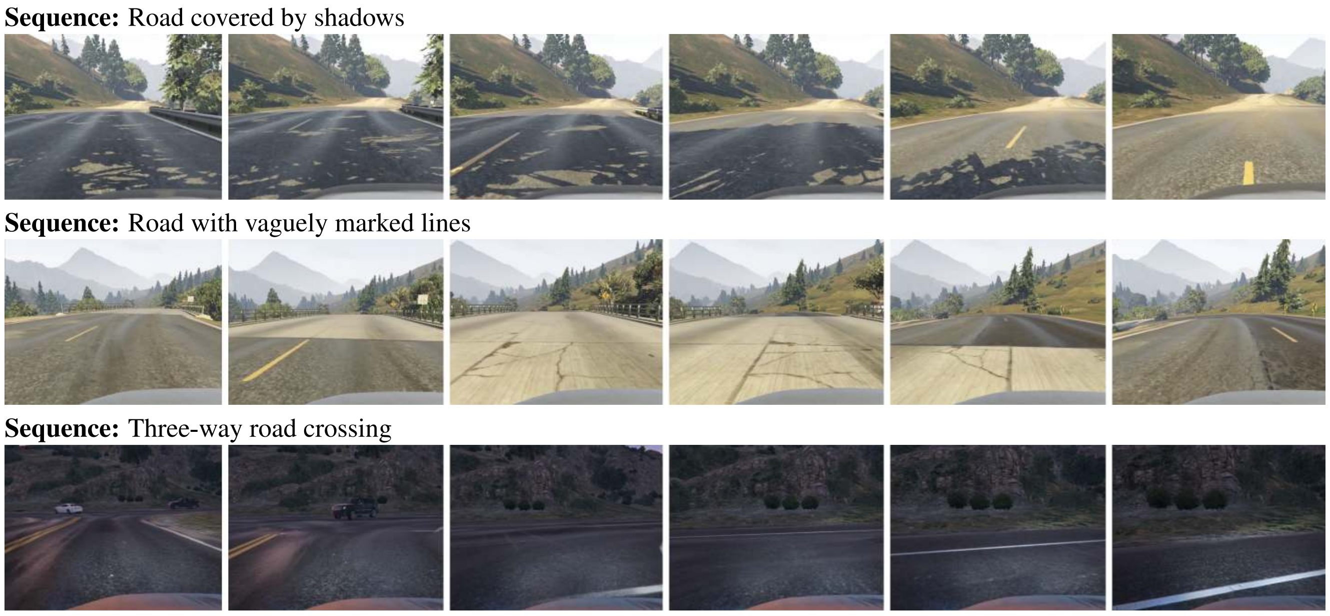
Sequence: Road covered by shadows
Sequence: Road with vaguely marked lines
Sequence: Three-way road crossing

Figure 18: Some video sequences showing the evaluated model driving in the GTA V simulator. We show some difficult cases: road covered by shadows (top), roads with vague lane markings, e.g. a bridge (middle), and a dark three-way crossing (bottom). Please refer to the supplementary material for the video version of these sequences.

ban environment, the network struggles with the same problem, resulting in poor real-time performance. Please refer to the supplementary material for videos depicting the cases discussed above.

Again, observations from this experiment suggest that current metrics are not always representative for real-time driving performance. In this regard, further research must be conducted towards developing new performance metrics and setting up automatic testing environments that are able to match performance at training time and performance during real-time driving. Some possible metrics could be distance from the middle of the lane, smoothness of driving (penalizing abrupt braking or turning), or a metric based on how long the car stays on the road without accidents.

5. Conclusion

In this paper, we analyzed an end-to-end neural network to predict the steering actions of a car on a highway from an input captured by a single car-mounted camera. Our analysis covered several high-level aspects of the neural network. These aspects were the format of the input data, the temporal dependencies between consecutive inputs and the application of simulated data.

Regarding the first aspect, we showed that the amount of classes of a classifier does not seem to have a big influence on the performance and that regression networks outperform classifier networks. This is likely due to the nature of their loss function which, similar to the metrics we use for evaluation, takes the magnitude of a prediction error into account. Moreover, we showed that, for the task at hand, there is no major difference between networks that use coloured images and ones that use grayscale images.

Regarding the second aspect, while we were unsuccessful in improving performance by implementing LSTM layers, the stacked frames approach delivered good results. By feeding the network 3 concatenated images, we got a significant decrease of 30% in mean square error (MSE). Further increasing the amount of concatenated images only brought diminishing returns that did not outweigh the drawbacks.

Regarding the third aspect, we were able to gather simulated data and train networks that have a performance comparable to the networks that we trained on real-life datasets. We have qualitatively shown the importance of recovery cases. We also qualitatively showed that the standard metrics that are used to evaluate networks that are trained on datasets - accuracy, MCA, MAE, MSE - do not necessarily reflect a system's driving behaviour. We have shown that a promising confusion matrix may result in poor driving behaviour while a very ill-looking confusion matrix may result in good driving behaviour. A structured framework is needed that allows to quantitatively measure more meaningful metrics. Finally, in future work we will explore domain adaptation and adversarial methods [27] as an alternative for adapting available data to the desired testing domain and further improve performance.

Acknowledgments: The authors thank Toyota for their generous support. This work was partially supported by the FWO SBO project OmniDrone, the KUL Grant PDM/16/131 and a NVIDIA Academic Hardware Grant.

References

[1] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[2] T. P. Cao and G. Deng. Real-time vision-based stop sign detection system on fpga. In *Computing: Techniques and Applications, 2008. DICTA'08. Digital Image*, pages 465–471. IEEE, 2008.

[3] C. Chen, A. Seff, A. Kornhauser, and J. Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

[4] F. Chollet et al. Keras. `https://keras.io/layers/recurrent/\#lstm`, 2015. [Online; accessed 08-May-2017].

[5] J. M. Collado, C. Hilario, A. De la Escalera, and J. M. Armingol. Model based vehicle detection for intelligent vehicles. In *Intelligent Vehicles Symposium, 2004 IEEE*, pages 572–577. IEEE, 2004.

[6] P. Dollar, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE transactions on pattern analysis and machine intelligence*, 34(4):743–761, 2012.

[7] F. A. Gers and J. Schmidhuber. Recurrent nets that time and count. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 189–194. IEEE, 2000.

[8] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610, 2005.

[9] C. Hilario, J. M. Collado, J. M. Armingol, and A. de la Escalera. Pedestrian detection for intelligent vehicles based on active contour models and stereo vision. In *International Conference on Computer Aided Systems Theory*, pages 537–542. Springer, 2005.

[10] S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[11] S. Hochreiter and J. Schmidhuber. Lstm can solve hard long time lag problems. In *Advances in neural information processing systems*, pages 473–479, 1997.

[12] M. Johnson-Roberson, C. Barto, R. Mehta, S. N. Sridhar, K. Rosaen, and R. Vasudevan. Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks? In *IEEE International Conference on Robotics and Automation*, pages 1–8, 2017.

[13] K. Kelchtermans and T. Tuytelaars. How hard is it to cross the room?–training (recurrent) neural networks to steer a uav. *arXiv preprint arXiv:1702.07600*, 2017.

[14] J. Kim and J. Canny. Interpretable learning for self-driving cars by visualizing causal attention. *arXiv preprint arXiv:1703.10631*, 2017.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[16] Y. LeCun, U. Muller, J. Ben, E. Cosatto, and B. Flepp. Off-road obstacle avoidance through end-to-end learning. In *NIPS*, pages 739–746, 2005.

[17] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 163–168. IEEE, 2011.

[18] H. Liu and B. Ran. Vision-based stop sign detection and recognition system for intelligent vehicles. *Transportation Research Record: Journal of the Transportation Research Board*, (1748):161–166, 2001.

[19] D. Neven, B. D. Brabandere, S. Georgoulis, M. Proesmans, and L. V. Gool. Fast scene understanding for autonomous driving. In *IEEE Symposium on Intelligent Vehicles*, 2017.

[20] OpenAI. Universe, measurement and training for artificial intelligence. `https://universe.openai.com/`. [Online; accessed 19-May-2017].

[21] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena. From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. *arXiv preprint arXiv:1609.07910*, 2016.

[22] D. A. Pomerleau. Alvinn, an autonomous land vehicle in a neural network. Technical report, Carnegie Mellon University, Computer Science Department, 1989.

[23] C. Quiter. deepdrive.io gta v dataset. `https://archive.org/details/deepdrive-baseline-uint8`.

[24] Rockstar Games Inc. Grand Theft Auto V, 2015.

[25] F. Sadeghi and S. Levine. $(cad)^2$ rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.

[26] E. Santana and G. Hotz. Learning a driving simulator. *arXiv preprint arXiv:1608.01230*, 2016.

[27] E. Santana and G. Hotz. Learning a driving simulator. *CoRR*, abs/1608.01230, 2016.

[28] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[29] Z. Sun, G. Bebis, and R. Miller. On-road vehicle detection: A review. *IEEE transactions on pattern analysis and machine intelligence*, 28(5):694–711, 2006.

[30] M. Sundermeyer, I. Oparin, J.-L. Gauvain, B. Freiberg, R. Schlüter, and H. Ney. Comparison of feedforward and recurrent neural network language models. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8430–8434. IEEE, 2013.

[31] M. Sundermeyer, R. Schlüter, and H. Ney. Lstm neural networks for language modeling. In *Interspeech*, pages 194–197, 2012.

[32] Udacity. Self-driving car simulator. `https://github.com/udacity/self-driving-car-sim`, 2017.

[33] B. Völz, K. Behrendt, H. Mielenz, I. Gilitschenski, R. Siegwart, and J. Nieto. A data-driven approach for pedestrian

intention estimation. In *Intelligent Transportation Systems (ITSC), 2016 IEEE 19th International Conference on*, pages 2607–2612. IEEE, 2016.

[34] H. Xu, Y. Gao, F. Yu, and T. Darrell. End-to-end learning of driving models from large-scale video datasets. *arXiv preprint arXiv:1612.01079*, 2016.