End-To-End Imitation Learning of Lane Following Policies Using Sum-Product Networks

Renato Lui Geh[1], **Denis Deratani Mauá**[1]

[1]Institute of Mathematics and Statistics – University of São Paulo (USP)

`{renatolg,ddm}@ime.usp.br`

Abstract. *Recent research has shown the potential of learning lane following policies from annotated video sequences through the use of advanced machine learning techniques. They however require high computational power, prohibiting their use in low-budget projects such as educational robotic kits and embedded devices. Sum-product networks (SPNs) are a class of deep probabilistic models with clear probabilistic semantics and competitive performance. Importantly, SPNs learned from data are usually several times smaller than deep neural networks trained for the same task. In this work, we develop an end-to-end imitation learning solution to lane following using SPNs to classify images into a finite set of actions. Images are obtained from a monocular camera, which is part of the low-cost custom made mobile robot. Our results show that our solution generalizes training conditions with relatively few data. We investigate the trade-off between computational and predictive performance, and conclude that sacrificing accuracy for the benefit of faster inference results in improved performance in the real world, especially in resource constrained environments.*

1. Introduction

Autonomous driving has gained traction in recent years thanks to many advances in both computer vision and machine learning. An important task in autonomous driving is lane following, where the self-driving agent must act so as to remain inside lane boundaries. Most approaches to lane following require accurate state estimations and on-line planning [Levinson et al. 2011, Paden et al. 2016, Pan et al. 2018], whose financial and computational costs can be prohibitive for low-budget projects, such as educational robotics and embedded devices. Recent work however has shown the potential of end-to-end imitation learning of lane following policies from monocular images as a viable and cost-effective alternative [Bojarski et al. 2016, Chen and Huang 2017, Pan et al. 2018]. While these works show impressive results, they rely on very large deep neural networks, which are difficult to deploy to resource-constrained agents.

Sum-product networks (SPNs) are deep probabilistic graphical models capable of representing complex probability distributions over large domains. SPNs have achieved impressive results in various traditional computer vision tasks, such as image completion [Poon and Domingos 2011, Dennis and Ventura 2012] and image classification [Gens and Domingos 2012, Sguerra and Cozman 2016, Peharz et al. 2018]. Exact inference in SPNs is linear in the size of its computation graph, making them an attractive alternative to deep neural networks for real-time image classification. The inherent probabilistic semantics of SPNs allow for efficient structure learning [Dennis and Ventura 2012, Gens and Pedro 2013, Vergari et al. 2015] and facilitates debugging. SPNs can also take advantage of the advanced optimization toolset of deep neural networks [Poon and Domingos 2011, Gens and Domingos 2012, Zhao et al. 2016b, Rashwan et al. 2018], to obtain competitive performance [Peharz et al. 2018].

In this work, we present an end-to-end learning approach for lane following in low-performance low-cost vehicles that uses SPNs to classify images from monocular cameras into actions in real-time. We develop several SPN classifiers by combining known structure and parameter learning algorithms. Importantly, we evaluate our system on a low-cost custom made mobile robot, under different lighting and floor conditions. Our results show that, even in contrived environments, the ability to make timely decisions is often more important than the accuracy of such decisions, and that less accurate but faster policies display better behavior than more accurate but slower policies.

This document is organized as follows. We start in Section 2 with a review of the recent literature on imitation learning applied to autonomous driving, and a brief description of our approach. We then provide in Section 3 the necessary background on SPNs. We describe in detail our approach in Section 4, and discuss its empirical evaluation in Section 5. Section 6 concludes the paper.

2. End-To-End Learning For Lane Following

In imitation learning, the agent learns a policy (i.e., a mapping from percepts to actions, a.k.a. a controller) for a given task from a sequence of observations of a teacher, usually a human expert [Hussein et al. 2017]. Thus, imitation learning reduces the problem of planning in complex environments to supervised learning, where one estimates a function from input-output pairs.

Possibly the first use of imitation learning for autonomous driving traces back to the seminal work of Pomerleau [Pomerleau 1989], who designed a self-driving agent for road driving based on a neural network policy that operated on monocular images and distance sensor measurements. Most subsequent works adopted a model-plan-act approach, which constructs policies on-the-fly, as needed [Levinson et al. 2011, Paden et al. 2016].

Recently, there has been a surge of interest in revisiting imitation learning for self-driving, especially for high-speed and/or resource-constrained agents. Particularly, there has been several proposals built around the idea of end-to-end learning where a steering policy is learned directly from percept input (e.g. raw image data), with very few pre- and post-processing on data [Pfeiffer et al. 2017, Chen and Huang 2017, Pan et al. 2018].

[Bojarski et al. 2016] trained a convolutional neural network from annotated video footage obtained from three cameras attached to a car to generate a policy that maps raw pixels to steering wheel angles. [Chen and Huang 2017] followed a similar approach using a single monocular camera. [Pan et al. 2018] combined imitation learning and reinforcement learning to obtain a steering policy that takes data from a front camera and wheel speed sensors. [Sguerra and Cozman 2016] employed imitation learning to learn steering policies for aerial mobile vehicles that need to operate at high speeds and with minimum weight; the latter requirement implies in a minimal sensoring set up. Similarly to this work, they used a sum-product network to classify images into a finite set of actions, and reported satisfactory performance on preliminary results. Their system however focused on optimizing accuracy and did not consider the computational cost of real-time decision making.

[Moraes and Salvatore 2018] adapted the approach of [Bojarski et al. 2016] to operate on a very resource-constrained agent built using educational robotic kits and custom made hardware. They trained deep neural networks for classifying images obtained from a low-cost monocular camera into three actions (move left, forward or right). They also did not consider the interplay of accuracy and inference speed, and their effect on real-time decision making.

In this work, we combine ideas from [Sguerra and Cozman 2016] and [Moraes and Salvatore 2018], and cast lane following policy construction as an image classification problem, where the goal is to learn a model that predicts one of three available actions (move left, forward or right) given an image. Following [Sguerra and Cozman 2016], we learn a sum-product network representing a *probabilistic policy*, that outputs a probability distribution over the actions, from which the agent selects the action with highest probability. We use the same hardware setup used in [Moraes and Salvatore 2018]. Image processing and classification is performed on a Raspberry Pi 3 Model B mounted on a three-wheeled mobile robot. Once the action is selected, a message is sent to the Lego Mindstorms NXT processing unit, which translates the action into power output signals to motor actuators. Figure 1 shows the robot's camera feed with the overlayed policy and the assembled low-cost mobile robot.

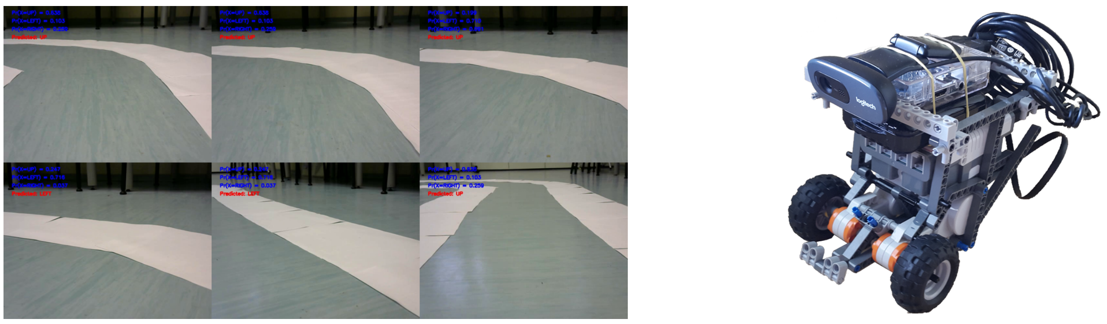

Figure 1. Left: lane following camera feed overlayed with policy action. Right: mobile robot used in our experiments.

3. Sum-Product Networks

In this work, we use a sum-product network (SPN) to represent a sophisticated probabilistic policy. An SPN represents a tractable probability distribution over a set of random variables (RVs) $\mathbf{X} = \{X_1, \dots, X_n\}$, called its *scope*, as a hierarchical mixture model. SPNs can be defined inductively, as follows.

Definition 1. *An SPN is either*

1. *a univariate probability distribution; or*
2. *a product of SPNs with disjoint scopes; or*
3. *a weighted sum of SPNs with identical scopes and nonnegative weights.*

SPNs are usually graphically represented by their computation graph, which is a rooted directed acyclic graph whose internal nodes are products and weighted sums, and whose leaves are random variables associated to univariate distributions. Figure 2 depicts

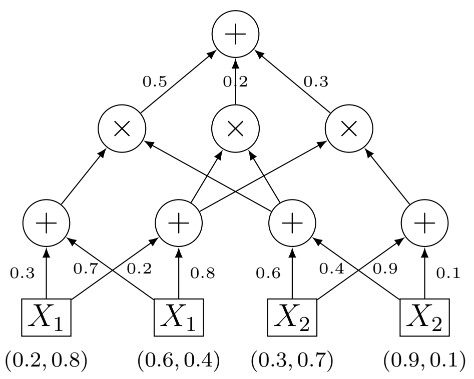

Figure 2. A sum-product network over Bernoulli random variables.

an SPN whose scope are Bernoulli random variables X_1 and X_2. The numbers underneath each leaf indicate the corresponding probability distribution.

The (unnormalized) probability $S(\mathbf{x})$ defined by an SPN S for a given configuration $\mathbf{x}$ of its scope can be computed recursively from the values of subnetworks as follows. Denote by S_n the SPN rooted at a node n in the DAG of an SPN S, and by $\text{Ch}(n)$ the set of its children. The value of a leaf node n is the probability value $P_n(\mathbf{x})$ of the associated univariate distribution at the respective value in $\mathbf{x}$. The value of an internal node n is given by $S_n(\mathbf{x}) = \prod_{j\in\text{Ch}(n)} S_j(\mathbf{x})$, if n is a product node, and by $S_n(\mathbf{x}) = \sum_{j\in\text{Ch}(n)} w_{n,j}S_j(\mathbf{x})$, if n is a sum node, where $w_{n,j}$ is the weight associated with the edge from n to its children j. When the weights are normalized, that is, when $\sum_{j\in\text{Ch}(n)} w_{n,j} = 1$ for each sum node n, then $S(\mathbf{x})$ computes the exact joint probability value $P(\mathbf{x})$ [Poon and Domingos 2011]. In order to avoid duplicate computations, the value of $S(\mathbf{x})$ is computed from the leaves toward the root, in a dynamic programming approach. For example, the probability of $X_1 = 1$ and $X_2 = 0$ specified by the SPN in Figure 2, computed by propagating values from the leaves towards the root, is 0.24.

If the weights are not normalized, then the probability of $\mathbf{x}$ is given by the ratio of $S(\mathbf{x})$ and Z, where Z is the value computed by the network when every leaf sends value 1 (which therefore can be computed efficiently).

An important feature of SPNs is the ability of computing the marginal probabilities in linear time, by a simple bottom-up propagation of values. The (unnormalized) probability of evidence $\mathbf{e}$ corresponding to a configuration of a subset of the variables in the scope, is obtained by the same recursive formulation as before with the following modification: The value of a leaf node n is 1 if X is not part of the evidence, otherwise it is the probability $P_n(\mathbf{e})$ specified by the associated univariate distribution. For example, the value of $P(X_1 = 1) = 0.5$ specified by the SPN in Figure 2, can be obtained by setting the values of leaves with scope X_2 to 1, and then performing the bottom-up propagation. Conditional probability values can be obtained with two passes over the network, hence also in linear time in the network size.

While marginal and conditional inference is tractable, computing the most probable configuration consistent with some evidence (a task known as MPE inference) is NP-hard in SPNs [Peharz 2015, Conaty et al. 2017, Mei et al. 2018]. Nevertheless, approximate algorithms, such as Max-Product [Poon and Domingos 2011] and Argmax-Product [Conaty et al. 2017], are known to produce reasonably accurate prediction, while maintaining efficiency. Max-Product, for instance, operates by replacing sums with maximizations in the computation of the value of sum and leaf nodes, and otherwise performing bottom-up propagation of values as in marginal inference. The corresponding configuration is extracted by backtracking the decisions of the maximizations, with a top-down pass in the network. Hence, the whole procedure still takes linear time in the size of the network.

3.1. Structure Learning

Several structure learning algorithms have been proposed to learn from data [Poon and Domingos 2011, Dennis and Ventura 2012, Vergari et al. 2015, Peharz et al. 2018, Gens and Pedro 2013]. In this work we consider the clustering architecture of Dennis and Ventura [Dennis and Ventura 2012], and different implementations of the LearnSPN schema [Gens and Pedro 2013].

Consider a dataset as a matrix indexed by variables on columns and values of variables as rows. The LearnSPN schema works by recursively partitioning data into clusters of rows, and sets of interdependent variables for columns. The schema takes the dataset and its scope as parameters, and at each step, LearnSPN checks if the dataset has column one, in which case a univariate distribution is created as a leaf. If that is not the case, it attempts to find partitions of its scope $P_1, \ldots, P_k$ such that for every pair of variable, X is independent of Y, where $X \in P_i$, $Y \in P_j$, $i \neq j$. This can be done through independence tests. A product node is created to represent this partitioning, with each of its children the recursive calls to LearnSPN under each P_i scope. If $k = 1$, i.e. the entire set is dependent, the algorithm creates a sum node with children as the recursive calls under the clusters $C_1, \ldots, C_m$ of the dataset as children. Each weight is then defined as the proportion $|C_i|/(|C_1| \cup \ldots \cup |C_m|)$. Figure 3 shows this recursive partitioning of the dataset.

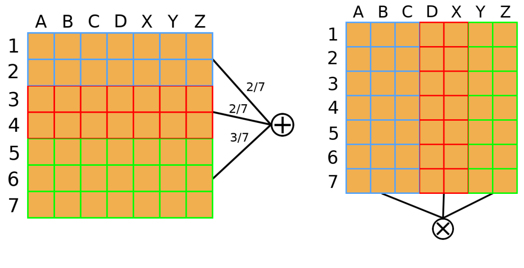

Figure 3. LearnSPN partition steps for sums and products.

The clustering architecture described in [Dennis and Ventura 2012], which we will refer to as the ClusterArch algorithm, works in a similar fashion to LearnSPN with some key differences. ClusterArch generates an SPN by k-means clustering both variables and their values. In other words, it uses clustering for both sums and products. In practice, this works by 2-clustering both rows and columns of the dataset, recursively partitioning the data into two. At each cluster step, instead of creating a single node, the algorithm creates a set of either sums or products. When the dataset has scope one, a mixture of gaussians is created.

3.2. Weight Learning

Weight learning in SPNs can take several forms [Poon and Domingos 2011, Gens and Domingos 2012, Rashwan et al. 2018, Zhao et al. 2016b, Zhao et al. 2016a]. For this work we explore generative and discriminative gradient descent and their soft and hard variants [Poon and Domingos 2011, Gens and Domingos 2012].

Poon and Domingos [Poon and Domingos 2011] showed how to train weights through generative gradient descent by taking the gradient of the log-likelihood and optimizing. This can be done through a weight update step

$$w_{n,j} \leftarrow w_{n,j} + \eta \frac{\partial S(\mathbf{x})}{\partial w_{n,j}},$$

where $\partial S / \partial w_{n,j}$ is the $n \to j$ edge component of the gradient and η the learning rate. Gens and Domingos [Gens and Domingos 2012] proposed discriminative gradient descent by computing the gradient of the conditional log-likelihood $P(\mathbf{y}|\mathbf{x})$, with weight update taking the form

$$w_{n,j} \leftarrow w_{n,j} + \eta \left(\frac{1}{S(\mathbf{y},\mathbf{x})} \frac{\partial S(\mathbf{y},\mathbf{x})}{\partial w_{n,j}} - \frac{1}{S(\mathbf{x})} \frac{\partial S(\mathbf{x})}{\partial w_{n,j}} \right).$$

These derivatives can be extracted through a single backpropagation-style top-down pass through the network.

A key issue of gradient descent in deep models is gradient diffusion, wherein the signal tends to fade the deeper the network. Both [Poon and Domingos 2011] and [Gens and Domingos 2012] attempt to solve this by using *hard* inference gradient descent. By using MPE instead of marginals, derivatives turn into constants. Each weight differential for the hard generative case turns into simple counts $w_{n,j} \leftarrow \eta \frac{c_{n,j}}{w_{n,j}}$, where $c_{n,j}$ is the number of times edge $n \to j$ appeared in Max-Product's top-down pass. Hard discriminative gradient descent works similarly, with each weight update taking the form $w_{n,j} \leftarrow \eta \frac{\Delta c_{n,j}}{w_{n,j}}$, with $\Delta c_{n,j}$ the difference between the counts for edge $n \to j$ in $S(\mathbf{y},\mathbf{x})$ and $S(\mathbf{x})$. Table 1 shows the weight update equations for each method.

Table 1. Gradient descent weight updates for SPNs.

GENERATIVE	
Soft	$\Delta w_{n,j} = \eta \frac{\partial S(\mathbf{x},\mathbf{y})}{\partial w_{n,j}}$
Hard	$\Delta w_{n,j} = \eta \frac{c_{n,j}}{w_{n,j}}$
DISCRIMINATIVE	
Soft	$\Delta w_{n,j} = \eta \left(\frac{1}{S(\mathbf{y},\mathbf{x})} \frac{\partial S(\mathbf{y},\mathbf{x})}{\partial w_{n,j}} - \frac{1}{S(\mathbf{x})} \frac{\partial S(\mathbf{x})}{\partial w_{n,j}} \right)$
Hard	$\Delta w_{n,j} = \eta \frac{\Delta c_{n,j}}{w_{n,j}}$

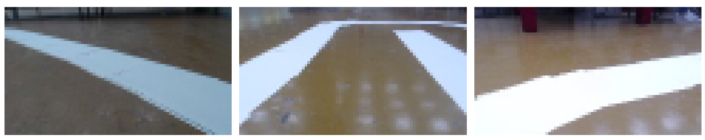
Figure 4. Samples from the training dataset: LEFT, UP and RIGHT, respectively.

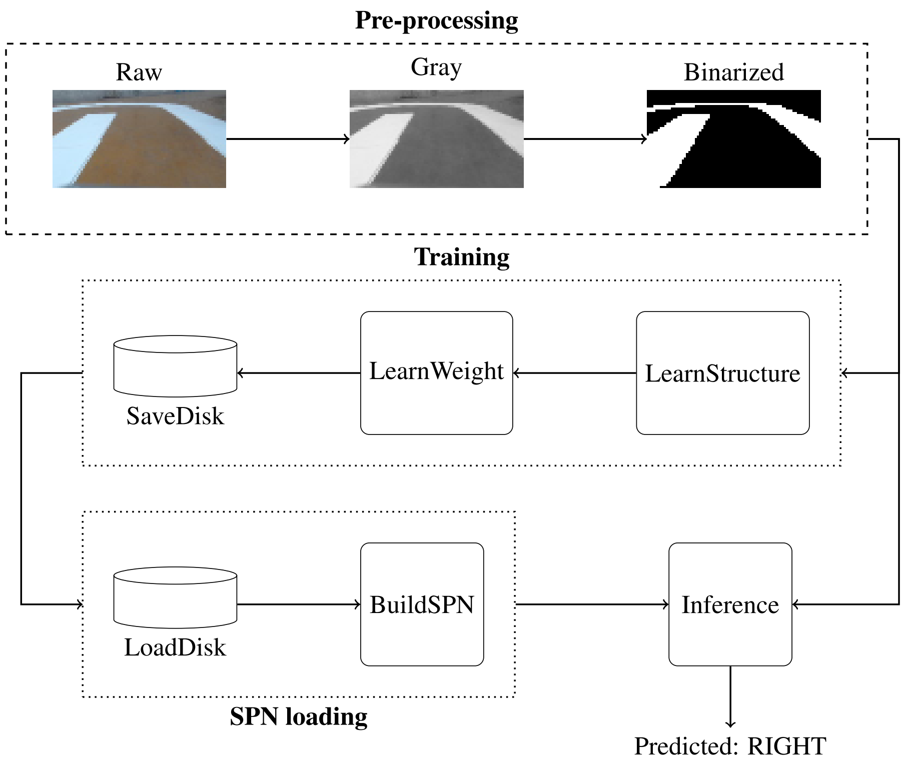

Figure 5. Pipeline for training and inference when using binarization.

4. Training SPNs for Lane Following

We applied image classification on Moraes and Salvatore's artificial self-driving dataset.[1] The dataset is composed of 56172 80×45 RGB lane following images labeled UP, LEFT and RIGHT, with labels representing actions taken by the human expert. Figure 4 shows samples for each label. The dataset was generated from a single track, and label distribution is uniform.

Images were converted from RGB to grayscale. An additional image transformation stage was then applied before training and prediction. Otsu's binarization [Otsu 1979], quantization, and equalization were used as possible pre-processing transformations. Figure 5 shows this pipeline. Pre-processing must be applied in real-time when predicting the agent's policy.

We trained structures with LearnSPN and ClassArch, with both hard generative and discriminative gradient descent for weights. We found that soft gradient descent yielded little to no improvement in accuracy. Derivatives were so small because of gradient diffusion, that weight updates were practically zero.

[1]Available at: `https://github.com/felipessalvatore/self_driving_data`

For LearnSPN, two variations using different methods for clustering were implemented. The first uses k-means with $k = 2$ and Euclidean distance as its metric, and the other DBSCAN [Ester et al. 1996]. Both apply the G-test for variable splitting. We found that, although DBSCAN with LearnSPN was able to achieve an impressive accuracy score of 100% on the test set; size and inference speed of this network was prohibitive for our mobile robot's limited hardware, as computing the agent's policy took almost 20 seconds for each frame, and its network's structure was 32 times larger compared to other trained SPNs. For these reasons, we omit results for this variant.

Table 2. Accuracy and prediction speed for each model

Accuracy (%)	**DV+g**	**DV+d**	**DV+s**	**GD+g**	**GD+d**	**GD+s**
B	78.8	78.8	78.8	82.8	83.8	85.0
Q_2	78.6	78.0	78.0	78.6	80.4	79.4
$Q_2 + E$	76.6	76.6	76.8	79.6	82.8	81.8
Q_3	77.4	77.4	77.4	77.6	80.2	79.8
$Q_3 + E$	70.4	76.6	76.6	79.2	81.2	77.4
Q_4	78.2	78.4	78.2	76.0	**78.2**	76.4
$Q_4 + E$	76.6	76.6	76.8	76.0	74.6	80.6
Q_5	77.8	78.4	78.4	77.6	74.0	73.8
$Q_5 + E$	76.6	76.6	76.6	72.0	72.8	72.0
Q_6	77.4	78.4	78.4	75.2	**74.4**	72.0
$Q_6 + E$	76.0	76.4	76.4	73.0	75.0	73.6
Q_7	78.2	78.4	78.4	62.8	72.2	71.4
$Q_7 + E$	76.2	76.4	76.4	70.6	71.4	71.6
$\emptyset$	78.0	78.4	78.4	62.4	**62.4**	62.4
E	76.4	76.4	76.4	60.4	60.0	61.2
Speed (secs)	**DV+g**	**DV+d**	**DV+s**	**GD+g**	**GD+d**	**GD+s**
B	0.23	0.25	0.25	0.38	0.37	0.31
Q_2	0.22	0.24	0.23	0.28	0.34	0.16
$Q_2 + E$	0.22	0.23	0.23	0.38	0.30	0.27
Q_3	0.22	0.23	0.22	0.22	0.32	0.17
$Q_3 + E$	0.22	0.23	0.22	0.34	0.32	0.31
Q_4	0.22	0.22	0.23	0.16	**0.17**	0.13
$Q_4 + E$	0.23	0.27	0.29	0.13	0.14	0.13
Q_5	0.22	0.26	0.28	0.07	0.05	0.02
$Q_5 + E$	0.22	0.29	0.25	0.05	0.05	0.02
Q_6	0.23	0.24	0.23	0.04	**0.05**	0.01
$Q_6 + E$	0.22	0.24	0.28	0.03	0.04	0.02
Q_7	0.23	0.23	0.26	0.03	0.01	0.01
$Q_7 + E$	0.22	0.26	0.24	0.01	0.01	0.01
$\emptyset$	0.22	0.26	0.23	0.02	**0.01**	0.01
E	0.23	0.23	0.22	0.01	0.01	0.02

All SPNs were trained with 500 samples of the 56172 total images of the dataset. This portion corresponds to only 0.9% of all available data. Another 500 were used for testing. When trained with 1000 samples, the SPNs generated were much more accurate, but inference speed decreased considerably. Both training and testing were run on a desktop computer. Table 2 shows test results for LearnSPN, referenced as **GD**; and ClassArch, referenced as **DV**. The letters **d** and **g** are used for indicating whether discriminative or generative gradient descent was used. We also trained ClassArch without optimizing weights by simply setting them as random, and LearnSPN through the proportional weights method discussed earlier. To distinguish such cases, an **s** is used. Pre-processing tasks are shown as B, for Otsu's binarization; Q_n, for n-bit quantization; and E for image equalization. An $\emptyset$ means images were trained with no image pre-processing.

5. Experiments and Results

Once all models were trained and evaluated on the test set, we picked three for testing on the custom-made mobile robot. These models correspond to the three bolded values in Table 2. We chose these models in order to evaluate performance on three cases: when using an accurate classifier with slower prediction speed, a fast model with poor accuracy, and a balanced middlepoint classifier.

As expected, we found that the difference between computation time on the desktop computer (an i7-4500U 1.80 GHz CPU), and the Raspberry Pi 3 Model B (Quad Core 1.2GHz Broadcom BCM2837 ARMv7) was quite significant. Table 3 shows the three chosen models with averages for their accuracy, desktop and mobile robot computation time for each prediction. We also show equivalent results for deep feed-forward neural networks (DFN) and convolutional neural networks (CNN) from [Moraes and Salvatore 2018]. Notice how, despite neural networks having much higher accuracy, inference takes twice as long at best, and 18 times longer at worst compared to SPNs.

Table 3. Accuracy and speed of models chosen for field testing

Model	**Accuracy (in %)**	**Desktop (in ms)**	**Robot (in ms)**
1: Q_4, GD+d	78.2	170	700
2: Q_6, GD+d	74.4	50	150
3: $\emptyset$, GD+d	62.4	10	75
DFN	81.3	-	$\approx$1350
CNN	80.6	-	$\approx$1350

Our approach to testing the trained models differ from Moraes and Salvatore's. In their work, movement is dependent on prediction, meaning the mobile robot only moves when the whole policy is readily available. This favors a more accuracy-based form of evaluation, as the agent may take as much time as it needs to decide on an action. We address the problem differently. In our implementation, the robot is always in motion. Once an action has been taken by the agent, it repeats itself until told otherwise. In this approach, prediction speed has a huge weight on performance, as an untimely decision might cause the vehicle to run off-track. This new environment mimics a more realistic scenario, as time sensitive decisions are often more relevant in the real-world. However, this new setting makes both the DFN and CNN models impractical, as a whole second is often too long to react to an incoming curve.

Three tracks were assembled for evaluation, as shown in Figure 6. Each of these tracks aimed to evaluate the classifier in different ways. The first is a simple square shaped circuit. The robot should ideally be able to smoothly navigate through both long straight lines and 90 degree angle curves. The second circuit is ∞-shaped, and sought to evaluate the model's ability of correctly identifying curves and lane intersections. The third and last track was a series of steep curves and narrow lines, simulating a road going down a mountain. Table 4 shows how each model performed on each circuit. The first and second columns refer to the classifier used and track run on. The third column indicates whether the agent was able to finish the track, i.e. whether it went out of bounds at any time. Column five shows the percentage of time spent within (but not on) lane markings. Video footage of the tracks and performance of each of the three policies is available online.[2] Results were evaluated by analyzing the footage of each model in each track.

Table 4. Performance of each model on the three tracks

Model	Track	Finished?	In
1: Q_4, GD+d	Track 0	Yes	82.0%
2: Q_6, GD+d		Yes	100%
3: $\emptyset$, GD+d		Yes	98.5%
1: Q_4, GD+d	Track 1	No	50.0%
2: Q_6, GD+d		Yes	97.0%
3: $\emptyset$, GD+d		Yes	69.6%
1: Q_4, GD+d	Track 2	No	44.4%
2: Q_6, GD+d		No	97.0%
3: $\emptyset$, GD+d		No	95.8%

Although Model 1 was the most accurate, we found that it was too slow to react when faced with new information. Model 3 performed better, but due to its low accuracy, often made mistakes that required further correcting. Having said that, it still achieved moderately good results, even though its movements were often spasmodic. Ultimately, Model 2 performed the best, as it provided a good balance between accuracy and latency. We emphasize the gaping difference between a fast but not so accurate policy versus an accurate but slow agent, noting that even though it often made mistakes, it still outperformed more accurate, slower models, as it had more chances to correct itself.

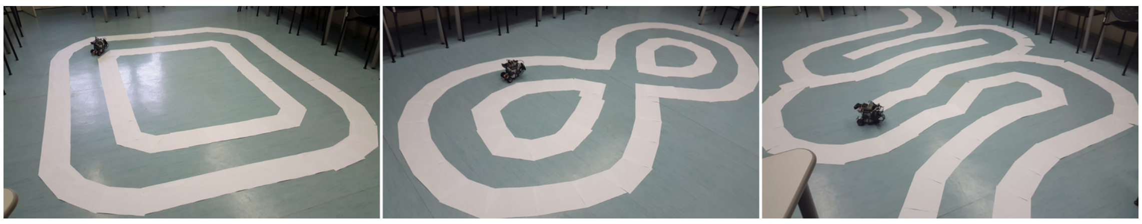

Figure 6. The three tracks used for evaluation.

The source code and data will be released as free and open software.[3]

6. Conclusions

We presented an end-to-end imitation learning approach for lane following policy generation in resource-constrained mobile robots. Our approach uses a sum-product network

[2]Available at: `https://youtu.be/vhpWQDX2cQU`.
[3]Available at: `https://github.com/RenatoGeh/godrive`.

to learn a sophisticated probabilistic policy from a dataset of annotated images. We show that sum-product networks achieve performance comparable to deep neural networks on this task, often with much smaller models. Our empirical results show that sacrificing classification accuracy for the sake of faster inference often leads to improved behavior. The results also show that sum-product networks learn effective policies, that generalize for different conditions such as lighting, floor color, and track topology. We leave as future work the evaluation in more realistic scenarios, such as city-like maps and under the presence of moving obstacles (e.g. other robots).

Acknowledgment

This work was supported by CNPq grant 133787/2019-2 and CAPES grant 88887.339583/2019-00. We would also like to thank Paula Moraes and Felipe Salvatore for useful discussion on the dataset and our implementation.

References

Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M. P., Muller, U., Zhang, J., Zhang, X., Zhao, J. J., and Zieba, K. (2016). End to end learning for self-driving cars. *CoRR*, abs/1604.07316.

Chen, Z. and Huang, X. (2017). End-to-end learning for lane keeping of self-driving cars. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1856–1860.

Conaty, D., de Campos, C. P., and Mauá, D. D. (2017). Approximation complexity of maximum A posteriori inference in sum-product networks. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence*.

Dennis, A. and Ventura, D. (2012). Learning the architecture of sum-product networks using clustering on variables. In *Advances in Neural Information Processing Systems 25*, pages 2033–2041. NIPS.

Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press.

Gens, R. and Domingos, P. (2012). Discriminative learning of sum-product networks. In *Advances in Neural Information Processing Systems 25*, pages 3239–3247. NIPS.

Gens, R. and Pedro, D. (2013). Learning the structure of sum-product networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 873–880, Atlanta, Georgia, USA. PMLR.

Hussein, A., Gaber, M. M., Elyan, E., and Jayne, C. (2017). Imitation learning: A survey of learning methods. *ACM Computing Surveys*, 50(2):21:1–21:35.

Levinson, J., Askeland, J., Becker, J., Dolson, J., Held, D., Kammel, S., Kolter, J. Z., Langer, D., Pink, O., Pratt, V., Sokolsky, M., Stanek, G., Stavens, D., Teichman, A., Werling, M., , and Thrun, S. (2011). Towards fully autonomous driving: Systems and algorithms. In *Proceedings of the IEEE Intelligent Vehicles Symposium*, pages 163–168.

Mei, J., Jiang, Y., and Tu, K. (2018). Maximum a posteriori inference in sum-product networks. In *AAAI Conference on Artificial Intelligence*.

Moraes, P. and Salvatore, F. (2018). Self-driving pi car. `https://github.com/felipessalvatore/self_driving_pi_car`.

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66.

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

Pan, Y., Cheng, C.-A., Saigol, K., Lee, K., Yan, X., Theodorou, E. A., and Boots, B. (2018). Agile autonomous driving using end-to-end deep imitation learning. In *Proceedings of Robotics: Science and Systems XIV*.

Peharz, R. (2015). *Foundations of Sum-Product Networks for Probabilistic Modeling*. PhD thesis, Graz University of Technology.

Peharz, R., Vergari, A., Stelzner, K., Molina, A., Trapp, M., Kersting, K., and Ghahramani, Z. (2018). Probabilistic deep learning using random sum-product networks. *CoRR*, abs/1806.01910.

Pfeiffer, M., Schaeuble, M., Nieto, J., Siegwart, R., and Cadena, C. (2017). From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1527–1533.

Pomerleau, D. A. (1989). Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems*, pages 305–313.

Poon, H. and Domingos, P. (2011). Sum-product networks: A new deep architecture. In *Proceedings of the Twenty-Seventh Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-11)*, pages 337–346, Corvallis, Oregon. AUAI Press.

Rashwan, A., Poupart, P., and Zhitang, C. (2018). Discriminative training of sum-product networks by extended baum-welch. In *Proceedings of the Ninth International Conference on Probabilistic Graphical Models*, volume 72 of *Proceedings of Machine Learning Research*, pages 356–367, Prague, Czech Republic. PMLR.

Sguerra, B. M. and Cozman, F. G. (2016). Image classification using sum-product networks for autonomous flight of micro aerial vehicles. In *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 139–144.

Vergari, A., Mauro, N. D., and Esposito, F. (2015). Simplifying, regularizing and strengthening sum-product network structure learning. In *ECML/PKDD*.

Zhao, H., Adel, T., Gordon, G., and Amos, B. (2016a). Collapsed variational inference for sum-product networks. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1310–1318. PMLR.

Zhao, H., Poupart, P., and Gordon, G. J. (2016b). A unified approach for learning the parameters of sum-product networks. In *Advances in Neural Information Processing Systems 29*, pages 433–441. NIPS.