Image Classification Using Sum-Product Networks for Autonomous Flight of Micro Aerial Vehicles

Bruno Massoni Sguerra
Escola Politécnica
Universidade de São Paulo, Brazil
Email: bmsguerra@gmail.com

Fabio Gagliardi Cozman
Escola Politécnica
Universidade de São Paulo, Brazil
Email: fgcozman@usp.br

***Abstract*—Flying autonomous micro aerial vehicles (MAVs) in indoor environments is still a challenging task, as MAVs are not capable of carrying heavy sensors as Lidar or RGD-B, and GPS signals are not reliable indoors. We investigate a strategy where image classification is used to guide a MAV; one of the main requirements then is to have a classifier that can produce results quickly during operation. The goal here is to explore the performance of Sum-Product Networks and Arithmetic Circuits as image classifiers, because these formalisms lead to deep probabilistic models that are tractable during operation. We have trained and tested our classifiers using the Libra toolkit and real images. We describe our approach and report the result of our experiments in the paper.**

I. Introduction

Unmanned aerial vehicles (UAVs) have demonstrated enormous potential in a wide range of applications, such as rescue missions, monitoring, research and exploration. While there is a significant effort to produce fully autonomous UAVs, there are still challenges when it comes to flying autonomous Micro Aerial Vehicles (MAVs) that are between 0.1-0.5 meters in length and 0.1-0.5 kilograms in mass [1] .

Much of the progress made in UAV autonomous flight in indoor environments is based on solutions such as Simultaneous Localization and Mapping (SLAM) or other feature tracking algorithms as described in [2]. However, SLAM is not a viable solution for MAVs due to the fact that building a 3D model is computationally heavy [3] and the use of feature extraction techniques does not perform well in environments devoid of trackable features such as walls. Other traditional solutions often involve the use of GPS or, alternatively, the use of sensors such as Lidar [4] or RGB-D [5]. However, GPS shows limited precision in indoor environments. Moreover, a MAV's reduced dimensions make it incapable of carrying heavy sensors, thus restricting it to lightweight sensors such as cameras. Indeed, the use of video cameras in such tasks has shown great promise [6].

The application of an image classifier in real-time autonomous flight using deep learning models such as deep convolutional networks is a practical solution to enable a quadcopter to navigate autonomously indoors environments [3]. However, MAVs cannot budget much processing time for the relatively complex task of image classification. The development of fast classifiers is crucial here.

In this paper we develop image classifiers that receive indoor images and classify them with respect to the actions that a flying MAV must take. Our classifier are based on Sum-Product Networks (SPNs), a "deep architecture" introduced by Poon and Domingos [7]. SPNs have the ability to encode probability distributions in a *tractable* form (that is, the cost of probabilistic inference is polynomial in the size of the network). The SPNs we use are in fact combined with another tractable formalism, employing arithmetic circuits to encode mathematical expressions needed during classification. This mixture offers promising encodings for statistical classifiers and compares favorably to other models, such as Bayesian networks or Markov random fields, for which inference is quite costly. We pursue the idea that tractable inference is crucial to MAV flying motion in developing our image classifiers.

Our long term goal is to simulate a pilot's choice of action using a Sum-Product Network (coupled with arithmetic circuits as needed). Our goal in this paper is more focused: we are interested in evaluating the performance of SPNs as a viable deep architecture for image classification. To train the networks we use, we resort to a dataset composed of images of an indoor environment and the corresponding pilot's commands. Our training and testing datasets consist of such images and actions.

We review needed concepts in Section II. In Section III we present our classification strategy, and in Section III-D we describe our experiments and results. Section IV summarizes our results and discusses future work.

II. Background

In this section we collect needed concepts. Section II-A describes Sum-Product Networks, and Section II-B describes the related language of arithmetic circuits. Section II-C reviews the learning algorithm we have used to produce our classifiers as well as the classification tests.

A. Sum-Product Networks

An important limitation of classical probabilistic graphical models, such as Bayesian networks and Markov networks, is the fact that inference in these models is intractable: computing the exact probability of a marginal or conditional query is $\#P - complete$ [8]. Therefore, if a network is large and has many connections, we cannot expect to run inference

978-1-5090-3566-3/16 $31.00 © 2016 IEEE
DOI 10.1109/BRACIS.2016.25

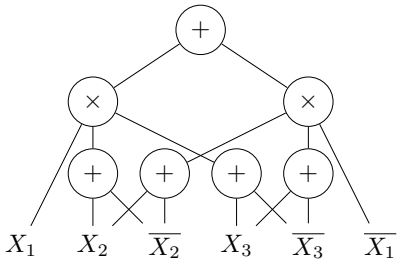

Fig. 1. Sum-Product Network.

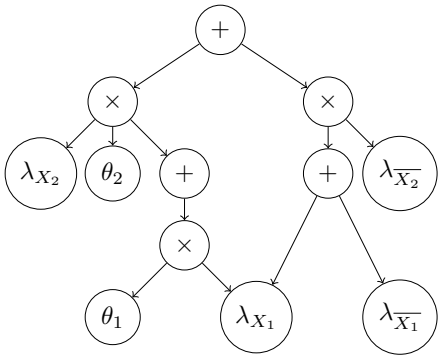

Fig. 2. Arithmetic Circuits.

quickly (where "inference" is the calculation of a conditional probability). Moreover, the difficulty may scale exponentially with the number of variables in the model.

Poon and Domingos have introduced Sum-Product networks as a model with tractable inference, as the cost of inference for a network is polynomial in the size of the network. Thus we can hope to quickly obtain exact probabilities in a practical problem that requires fast response (such as MAV flying control). SPNs offer a deep architecture in the sense that they are built in layers that can be indefinitely stacked.

Similarly to classical probabilistic graphical models, SPNs are graphical description of probability distributions. Basically, SPNs are rooted, directed, acyclic graphs composed by sum and products as internal nodes, with variables as leaves, and positive weighted edges (see Figure 1). There are two interpretations for SPNs. One, as we have advanced already, as tractable probabilistic graphical models. Another interpretation is that they are deep neural networks with two types of neurons: sums and products [9].

To define a SPN, we can, for simplicity, focus on Boolean variables, as the extension to multi-valued discrete variables and continuous variables is simple. The indicator function [.] of a Boolean variable has value 1 when the argument is true, and value zero otherwise. We will abbreviate $[X_i]$ (the indicator function of variable X_i) by x_i and $[\overline{X_i}]$ by $\overline{x_i}$. A SPN is denoted as a function S of the indicator variables $x_1, ..., x_N$ and $\overline{x_1}, ..., \overline{x_N}$ by S($x_1, ..., x_N$,$\overline{x_1}, ..., \overline{x_N}$). When the indicators specify a complete state x we abbreviate it as $S(x)$, when they specify evidence e, we abbreviate it as $S(e)$.

The following two definitions capture the main concepts:

Definition 1: A SPN over finite state random variables $X = \{X_1, X_2, ..., X_N\}$, is a rooted acyclic graph whose leaves are indicators $x_1, ..., x_N$ and $\overline{x_1}, ..., \overline{x_N}$, whose internal nodes are sum and products. Each of the edges (i, j) out of a sum node has a non-negative weight associated with it $w_{i,j}$. The value of a product node is the product of its children values, and the value of a sum node is $\sum_{j \in Ch(i)} w_{i,j} v_i$, where $Ch(i)$ are the children of i and vj is the value of node j. The value of a SPN is the value of its root [7].

Definition 2: An unnormalized probability distribution $\Phi(x)$ is representable by a sum-product network S iff $\Phi(x) = S(x)$ for all states x and S is valid (a SPN is valid if it is correct at computing the probability of evidence)[7].

S therefore computes correctly all marginals of $\Phi(x)$ including its partition function.

From these definitions, one can prove that the partition function of a Markov network $\Phi(x)$, where x is a d-dimensional vector, can be computed in time polynomial in d if $\Phi(x)$ is representable by a SPN with a number of edges polynomial in d [7].

B. Arithmetic Circuits

Arithmetic circuits (ACs) [10] are closely related to SPNs; they are rooted, directed, acyclic graphs with sum and products as internal nodes (see Figure 2). However, while SPNs have univariate distributions as leaves, ACs have indicator nodes and parameter nodes as leaves [8].

There are tree important properties of ACs:

- an AC is decomposable if the children of a product node have disjoint scopes.
- an AC is deterministic if the children of a sum node are mutually exclusive, meaning that at most one is non-zero for any complete configuration.
- an AC is smooth if the children of a sum node have identical scopes.

If an AC is smooth and decomposable, then it represents a valid probability distribution.

ACs and SPNs can be equivalent; it has been shown that, for discrete domains, every decomposable and smooth AC can be represented by a SPN with fewer or equal number of nodes and edges. Moreover, again in discrete domains, every SPN can be represented by an AC with at most a linear increase in the number edges [8].

C. Training Classifiers and Computing Inferences with the Libra Toolkit

We have used the **ID-SPN** algorithm to learn our classifiers from data, and the **spquery** algorithm to compute inference and test our classifiers. These algorithms are implemented in the Libra Toolkit, a collection of algorithms put together by Lowd and Rooshenas [11] and geared towards learning and computing inference with a series of probabilistic models. In this section, we present a brief overview of these two algorithms.

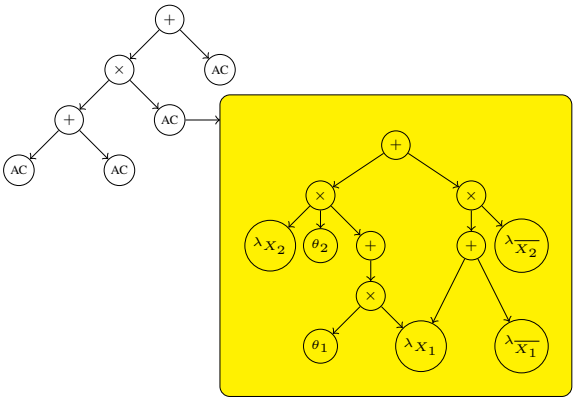

Fig. 3. An ID-SPN model. The upper level variables are shown explicitly as sum and product nodes, while the lower level nodes represented by AC are arithmetic circuits over the observed variables.

For both algorithms, Libra uses comma-separated lists of variable values as datasets, with asterisks representing values that are unknown. This allows the same format to be used for training examples and evidence configurations. A sequence such as $0, 0, 1, 0, 4, 3, 0$ is a valid datapoint. Another valid sequence is $0, 0, 1, *, 4, 3, *$, this time with a missing value.

*1) The **ID-SPN** algorithm:* The ID-SPN algorithm is used to learn SPNs using *direct* and *indirect* interactions.

In order to learn SPNs, the ID-SPN algorithm uses top-down clustering to detect direct interactions; that is, it partitions the training data to create a sum node (that represent a mixture over different clusters) and it partitions the variables to create a product node (that represents groups of independent variables within a cluster). However, since this method has difficulty in finding direct interactions among variables, ID-SPN then combines it with methods for learning tractable Markov networks represented as ACs.

Therefore, ID-SPN looks for indirect interactions through latent cluster variables in the upper levels of the SPN, as well as direct interactions through the tractable Markov networks at the lower levels of the SPN. Traditionally, to learn a SPN, most algorithms uses top-down clustering until it reaches univariate distributions. ID-SPN, however, may choose to stop this process before reaching univariate distributions and instead learn an AC to represent a tractable multivariate distribution with no latent variables.

An alternative way to understand ID-SPN is that it learns SPNs where the leaves are tractable multivariate distributions rather than univariate distributions. As long as these leaf distributions can be represented as valid SPNs, the overall structure can be represented as a valid SPN as well [8].

We refer to the structure learned by the ID-SPN algorithm as a sum-product of arithmetic-circuits, or SPAC for short (see Figure 3).

Concerning the Libra Toolkit, there are several parameters that control the behavior of the ID-SPN implementation; relevant parameters are given in Table I, together with their default values.

TABLE I
ID-SPN PARAMETERS.

l1	Weight of L1 norm [5.0]
l	EM clustering penalty [0.2]
ps	Per-split penalty [10.0]
k	Max sum nodes' cardinalities [5]
sd	Standard deviation of Gaussian weight prior [1.0]
cp	Number of concurrent processes [4]
vth	Vertical cut threshold [0.001]
ext	Maximum number of node extensions [5]
minl1	Minimum value for components' L1 priors [1.0]
minedge	Minimum edge budget for components [200000]
minps	Minimum split penalty for components [2.0]
seed	Random seed
f	Force to override output SPN directory

In this paper, we set the parameters cp, l and vth to 2, 0.2 and 0 respectively, and we focused on varying the values of k (which control the number of sum nodes; this parameter is related to the partition of our training data), ext (which, if large, may cause the network to take a long time to train and may cause overfitting), ps and $l1$ in order to analyze the different learned classifiers.

*2) The **spquery** algorithm:* The **spquery** algorithm is used to compute exact inference in SPNs.

There are three inputs to this algorithm, the SPN (SPAC) model directory containing the learned SPN, an evidence file and a query file.
Evidence files have the same comma separated format used for the data file. When the value of a variable has not been observed as evidence, we put a "*" in the related column:

$$0, 0, 1, *, 4, 3, *$$

The query file must be consistent with the evidence. For example:

$$0, 0, 1, 1, 4, 3, 0$$

For instance, the **spquery** method can be used with the above query and evidence to find the conditional probability of: $P(X_4 = 1, X_7 = 0 | X_1 = 0, X_2 = 0, X_3 = 1, X_5 = 4, X_6 = 3)$.

III. DEVELOPMENT

A. Overview

As noted in the Introduction, our goal is to learn an image classifier using a SPAC, in order to obtain decisions concerning actions during a MAV flight in indoor environments. Our classifier is trained through a dataset consisting of different images of indoor hallways and the corresponding human pilot's choice of action. The classifier works by computing the probability of an image being part of each class, and returning the class with the higher probability. In this paper we focus on classifier training using collected images.

In future work, we intend to use the trained classifier in a flying MAV; our trained classifier will input real-time images

taken from the MAV and return a flight command that best mimics a human pilot's behavior.

B. Data: images and actions

Our dataset was build using captured images of several different indoor locations as shown in Fig. 4.

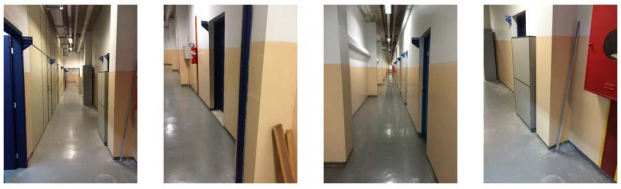

Fig. 4. Sample of images from our dataset.

Every image was classified as a MAV command corresponding to a human pilot's choice of action. In this paper, we focus on three classes of images: "move forward", "move left" and "move right". In Fig. 5 we have some sample images and their corresponding command.

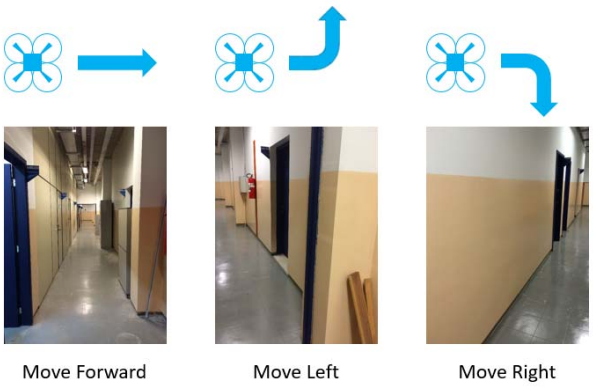

Fig. 5. Sample images with their corresponding command.

Each data point corresponds to one image of our dataset. Libra's current implementation of ID-SPN only supports binary-valued variables; therefore, we had to transform our pictures into binary images. This was done by selecting a value of luminance level as threshold and substituting all the pixels with luminance under the threshold by 0 (black) and the others by 1 (white). Using a threshold of 0.45, we obtained black and white images as shown in Fig. 6.

We then reshaped our images to 1% of their original size, as shown in Fig. 7. By doing so, we lost some information, but the reduction in size allowed ID-SPN to train our networks much faster, each network taking approximately 5 minutes to train.

To build our data file, we took each line of the black and white figure and concatenated them into a single one-dimensional array. Then, each array was stacked into our data file. At the end of each data point, we added two binary

Fig. 6. Black and white images obtained.

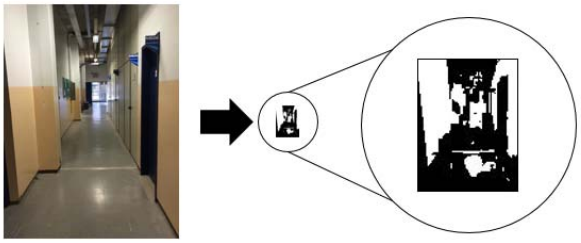

Fig. 7. Reduction to 0.01 scale.

variables representing our classes. For a classifier with just two classes, one single variable is needed. As we want to classify images into three classes, we had to add two variables corresponding to each class. 00 if it is a "move forward" image, 01 if it is a "move right" image and 10 if it is a "move left" image. Therefore, if we have a image with the following structure:

$$image_i = \begin{vmatrix} 1 & 0 & 1 & 0 & \dots & 1 \\ 0 & 1 & 0 & 1 & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & 1 & 1 & \dots & 0 \end{vmatrix},$$

our corresponding data point is then:

$$[1, 0, 1, 0, \dots, 1, |0, 1, 0, 1, \dots, 1, |\dots, |0, 1, 1, 1 \dots, 0, |1, 0],$$

where we added a | symbol to distinguish the various rows in the one-dimensional array. The last two values in this vector $1, 0$ indicate the corresponding class of the image as "move left".

To build our validation set, we used the **hold-out** cross-validation method, which is the simplest kind of cross-validation. We randomly separated our data into two sets, the dataset and the validation set. In Section III-D, to classify the images of our validation set we used the **spquery** method to compute the probability of an image being part of each class. The **spquery** method uses as input the evidence and query file (as well as the learned SPN). With the validation set defined we can build the evidence and the query files.

C. Evidence and Query Files

In order to build our evidence and query files, we employed the same procedure as used for building our data file. Each

image used to evaluate our classifier was transformed to black and white and reshaped to 1% of its size. Then each of the image's lines was concatenated into a single array.

The evidence file is composed of evidence points representing each image. As shown in the section II-C2, variables that has not been observed must be represented as "*", therefore we inserted two "*" at the end of each point indicating that we do not know the images class. As also shown in the section II-C2, in order to compute inference, there must be a corresponding query point to each evidence point.

The query file was then built using the same evidence points, but substituing the two "*" for the values corresponding to each of the possible classes.

Since we have three different classes, we need our evidence file to be composed of three equal evidence points for each image with a "**" at the end, this allows us to use the **spquery** method and compute the probability of an image being part of each of the three classes.

Therefore, for an image:

$$image_i = \left| \begin{array}{ccccc|} 0 & 0 & 0 & \dots & 0 \\ 1 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots \ \vdots \\ 1 & 0 & 1 & \dots & 1 \end{array} \right|$$

We create 3 evidence points and 3 query points:

evidence points:

$$\begin{aligned} e\,point_i &= [0,0,0,\dots,0,|1,0,1,\dots,0,|\dots,|1,0,1\dots,1,|*,*] \\ e\,point_{i+1} &= [0,0,0,\dots,0,|1,0,1,\dots,0,|\dots,|1,0,1\dots,1,|*,*] \\ e\,point_{i+2} &= [0,0,0,\dots,0,|1,0,1,\dots,0,|\dots,|1,0,1\dots,1,|*,*] \end{aligned}$$

query points:

$$\begin{aligned} q\,point_i &= [0,0,0,\dots,0,|1,0,1,\dots,0,|\dots,|1,0,1\dots,1,|0,0] \\ q\,point_{i+1} &= [0,0,0,\dots,0,|1,0,1,\dots,0,|\dots,|1,0,1\dots,1,|0,1] \\ q\,point_{i+2} &= [0,0,0,\dots,0,|1,0,1,\dots,0,|\dots,|1,0,1\dots,1,|1,0] \end{aligned}$$

And so we can compute the probablility, given the image as evidence, of each image being part of the "move forward", "turn right" or "move left" class.

D. Results

1) Binary Case: We start with a binary classification problem. That is, our dataset is composed only of images corresponding to two classes of commands "move right" or "move left". In this case, we used a dataset of 66 images and a validation set of 23 images. Using the procedure described before, we set the luminance (Lumi) threshold to 0.45 or 0.55 and built our data, evidence and query files.

To choose the *ID-SPN* parameters that results in the best classifier, we set $cp = 2$, $l = 0.2$ and $vth = 0$; then we generated random values for the parameters *k*, *ext*, *ps* and *l1*.

For each of the trained SPNs, we then used the *spquery* method to compute the probability of each one of our validation set images being part of class "move right" and class "move left". As indicated before, our classifier assigns to an image the class with the largest probability (in the case where the classifier returns the same probability for the image being part of both classes, it is considered an error). Therefore, knowing the actual class of an image, we can compute the accuracy (ACC) as

$$ACC = \frac{N_{Right} + N_{Left}}{N},$$

where N_{Right} is the number of instances where a "move right" image was correctly classified, N_{Left} is the number of instances where an image was correctly classified as "move left" and N is the total number of images in our validation set.

TABLE II
BINARY CLASSIFICATION

Classifier	Lumi	K	ext	ps	l1	Accuracy
1	0.55	44	3	6	6	75.00%
2	0.45	27	10	7	1	70.83%
3	0.45	30	5	5	11	66.70%
4	0.45	6	7	4	2	66.67%
5	0.55	30	6	20	20	62.50%
6	0.45	31	10	1	12	62.50%
7	0.45	38	9	9	15	62.50%
9	0.55	50	6	3	11	62.50%

Table II shows 9 of our trained SPNs, presented in descending order of accuracy, with classifier 1 being the most accurate with 75.00% of accuracy. This means that 75.00% of the pictures in our validation set were correctly classified.

2) Multiclass Classification: For our multiclass classifier our dataset is composed of images corresponding to the three classes "move forward", "move left" and "move right". In this case, we used a dataset of 93 images and a validation set of 33 images. We followed the same procedure used for building our binary data file. The evidence file and the query file were also built using the same procedure.

Once again we set $cp = 2$, $l = 0.2$, $vth = 0$ and used the values for the parameters *k*, *ext*, *ps* and *l1* that generated the SPNs with biggest accuracy in the binary case.

The accuracy in the multiclass case is defined, using the same notations as above, as $ACC = S/N$, where

$$S \;=\; N_{Forward} + N_{Right} + N_{Left}\ ,$$

where $N_{Forward}$ is the number of images correctly classified as "move forward". Table III shows 9 of the best trained classifiers and their respective values of accuracy.

TABLE III
MULTICLASS CLASSIFICATION

Classifier	Lumi	K	ext	ps	l1	Accuracy
1	0.45	41	10	9	6	72,73%
2	0.45	13	3	7	15	63.64%
3	0.45	11	6	9	1	60.61%
4	0.45	47	10	5	16	60.61%
5	0.45	23	8	9	10	60.61%
6	0.45	37	5	8	17	60.61%
7	0.45	59	6	5	18	60.61%
8	0.45	59	4	2	9	60.61%
9	0.45	31	10	9	12	57.58%

It is clear that the accuracy of our classifiers in the multiclass case is not as high as in the binary case. The most accurate classifier in the multiclass case could predict the class of 72,73% of our images. The top three classifiers shown in Table III (1,2 and 3) were very successful in classifying images corresponding to the command "move forward" with an average of 85.19% of accuracy. While they had problems classifying images corresponding to the command "move right" with an average of just 44.44% of accuracy. In classifying "move left", these classifiers presented an average of accuracy of 77.78%.

Therefore, there are a number of possibilities to enhance the effectiveness of our multiclass classifier. For instance, the parameters that worked best in the binary case may not be the best parameters for the multiclass case. In addition, the number of images in our data set might not be enough to train a more precise classifier. It is possible that, due to the cross-validation method used (hold-out), the images in the validation set might not be well represented in our dataset. Another possibility for enhancing our classifiers accuracy is increase the scale while reshaping the images used to build our dataset (Fig.7) and validation set. The scale used (0.01) to build our classifiers, although responsible for gains in the learning speed, caused information loss.

IV. Conclusion

The use of SPNs is promising for image classification in environments that require fast decision making. The advantage of SPNs (and SPACs) is that, even if training may be costly, inference during actual operation is tractable. Inference speed is crucial as we are interested in a real-time situation.

The Libra Toolkit offers powerful tools for learning and computing inference in SPN. Although Libra's interface is not optimal for image classification, with some pre-processing of the data it can produce classifiers in reasonable time.

Although our most accurate classifier had an accuracy of 75% (binary case), we obtained such result with a dataset of just 66 images. A larger dataset will certainly lead to more accurate classifiers. A better way of evaluating our classifiers is by applying a different cross-validation technique. The hold-out technique used in this paper, although simple and easy to implement, does not make optimal use of data, depending on which data points end up in the training set and which end up in the validation set. Techniques such as k-fold cross-validation may be more appropriate; although some data is also wasted and it is more expensive than the hold-out cross-validation, it is less affected by how the data are divided, and therefore is bound to obtain more accurate results.

Finally, in the autonomous flight solution proposed, there is only the need to learn a single SPN to be used as an image classifier. In this paper, since we were interested in training a number of different classifiers in order to evaluate their performance, learning speed was a relevant factor. If the time needed to learn the SPN (SPAC) is not a constraint, it is possible to obtain a more precise and complex SPN (SPAC) by increasing the scale (as shown in Fig.7) which will provide more information about the classes to the learning algorithm. Moreover, since the inference in SPNs is tractable, a more complex structure will not result in a significant decrease in the inference speed.

Future work should focus on training a more complex classifier, capable of classifying images into classes corresponding to more of the MAV's possible commands, like "spin right", "spin left", "go up, "go down", "stop", etc. Thus a dataset with images corresponding to all possible commands is also necessary. It is also necessary to implement the MAV-classifier interfaces; i.e., the connection between our MAV and our classifier, allowing the images captured by the MAV to be processed (resized to a smaller scale and turned binary) and sent to the classifier, and allowing the corresponding command given by the classifier to be sent back to the MAV in real-time. With these interfaces implemented, we can retrieve the image captured by the MAV at the moment the pilot perform each of his commands, and we can build our dataset in a dynamic context through a number of flight simulations with a human pilot. Finally, it involves testing our classifier with a MAV thought dynamic simulations.

Acknowledgment

We would like to thank Francisco Henrique Otte Vieira de Faria at the Decision Making Lab for useful discussions and help. Also, we would like to thank Guilherme Hatori Pereira Horoi for his help with the Libra toolkit.

References

[1] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2011, pp. 2520–2525.

[2] N. X. D. N. X. Dao, B.-J. Y. B.-J. You, S.-R. O. S.-R. Oh, and M. H. M. Hwangbo, "Visual self-localization for indoor mobile robots using natural lines," *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, vol. 2, no. October, 2003.

[3] D. K. Kim and T. Chen, "Deep neural network for real-time autonomous indoor navigation," 2015. [Online]. Available: http://arxiv.org/abs/1511.04668

[4] A. Bachrach, R. He, and N. Roy, "Autonomous flight in unknown indoor environments," *International Journal of Micro Air Vehicles*, vol. 1, no. 4, pp. 217–228, 2010.

[5] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Estimation, planning, and mapping for autonomous flight using an RGB-D camera in GPS-denied environments," *International Symposium on Robotics Research (ISSR)*, pp. 1–16, 2011. [Online]. Available: http://ijr.sagepub.com/cgi/doi/10.1177/0278364912455256

[6] R. Roberts, D.-N. Ta, J. Straub, K. Ok, and F. Dellaert, "Saliency detection and model-based tracking: a two part vision system for small robot navigation in forested environment," *SPIE Defense, Security, and Sensing*, pp. 83 870S–83 870S–12, 2012.

[7] H. Poon and P. Domingos, "Sum-product networks: A new deep architecture," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 689–690, 2011.

[8] A. Rooshenas and D. Lowd, "Learning sum-product networks with direct and indirect variable interactions," ... *International Conference on Machine Learning*, vol. 32, 2014. [Online]. Available: http://jmlr.org/proceedings/papers/v32/rooshenas14.html

[9] D. R. Peharz, "Foundations of Sum-Product Networks for Probabilistic Modeling," *PhD Thesis*, no. February 2015, 2015.

[10] A. Darwiche, "A differential approach to inference in Bayesian networks," *J. Acm*, vol. 50, no. 3, pp. 280–305, 2003.

[11] D. Lowd and A. Rooshenas, "The libra toolkit for probabilistic models," *Journal of Machine Learning Research*, vol. 16, pp. 2459–2463, 2015.