Training Decision Trees as Replacement for Convolution Layers

Wolfgang Fuhl, Gjergji Kasneci, Wolfgang Rosenstiel, Enkeljda Kasneci
Eberhard Karls Universität Tübingen
Sand 14
Tübingen, Germany
{wolfgang.fuhl, gjergji.kasneci, wolfgang.rosenstiel, enkelejda.kasneci}@uni-tuebingen.de

Abstract

We present an alternative layer to convolution layers in convolutional neural networks (CNNs). Our approach reduces the complexity of convolutions by replacing it with binary decisions. Those binary decisions are used as indexes to conditional distributions where each weight represents a leaf in a decision tree. This means that only the indices to the weights need to be determined once, thus reducing the complexity of convolutions by the depth of the output tensor. Index computation is performed by simple binary decisions that require fewer cycles compared to conventionally used multiplications. In addition, we show how convolutions can be replaced by binary decisions. These binary decisions form indices in the conditional distributions and we show how they are used to replace 2D weight matrices as well as 3D weight tensors. These new layers can be trained like convolution layers in CNNs based on the backpropagation algorithm, for which we provide a formalization.

Our results on multiple publicly available data sets show that our approach performs similar to conventional neuronal networks. Beyond the formalized reduction of complexity and the improved qualitative performance, we show the runtime improvement empirically compared to convolution layers.

Introduction and Related Work

Conditioning CNNs is a modern approach to reducing runtime which is typically achieved by activating only parts of the models or by pursuing the scalability of model complexity (Shazeer et al. 2017; Ioannou et al. 2016; Chen et al. 2018) to reduce computational costs without compromising accuracy. Recent approaches even reduce the complexity of convolution layers (Keskin and Izadi 2018) without affecting the accuracy. This paper describes a new approach for the practical implementation of conditional neural networks using conditional distributions and binary decisions. Similar to (Keskin and Izadi 2018), we replace convolutional layers to reduce computational complexity with the addition of indexing by simple binary decisions. We show analytically and empirically the reduction of the computational runtime on the basis of public data sets as well as the retention or increase of the accuracy of the model.

There are four main categories of Conditional Neural Networks:

1 Neural Networks that use loss functions for optimizing decision parameters.

2 Probabilistic approaches that learn a selection of experts.

3 Neural networks with decision tree architectures.

4 Replacement layers for the convolutions, which map hierarchical decision graphs conditionally to the input feature space.

The first category uses non-differentiable decision functions where the parameters for these are learned by an additional loss function. A loss function which maximizes the distances of the subcluster was presented in (Xiong et al. 2015). The path loss function is used in (Baek, Kim, and Kim 2017). This is based on the purity of the data activation with respect to its class label distribution. The information gain is used in (Bicici, Keskin, and Akarun 2018) to learn an evaluation function which allows to activate paths through the network.

In the second category, probabilistic approaches are pursued. Weights are assigned to each branch and treated as a sum over a loss function (Ioannou et al. 2016). A similar approach is followed in (Shazeer et al. 2017). The main difference is that a very high number of branches per layer is considered and the best k branches are followed in the training phase as well as in the test phase. Another approach trains two neural networks where one provides the decision probability at the output and the second network performs the classification (Kontschieder et al. 2015). Both nets are trained jointly.

In the third category, the architecture of the neural network is similar to a decision tree. Randomized multi-layer perceptrons are used in (Rota Bulo and Kontschieder 2014) as branch accounts and trained together with the entire net. An alternative architecture is presented in (Denoyer and Gallinari 2014). Here, each account in a net has three possible subsequent nodes. The selection of the following node is done via an evaluation function which is learned via the REINFORCE algorithm (Denoyer and Gallinari 2014). In (Wang, Aggarwal, and Liu 2017), partitioning features are learned which make it possible to train the whole network

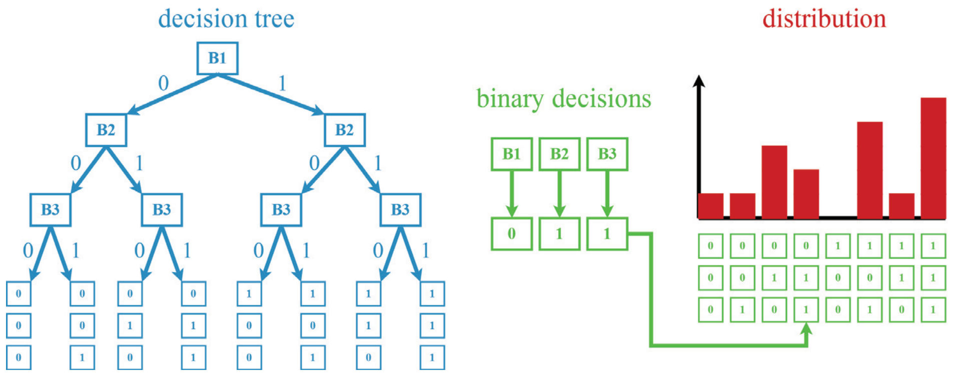

Figure 1: The left part depicts a decision tree. The decisions are true or false evaluations and the tree structure is complete and balanced. The middle part replaces the tree with a simple binary number and indexes a weight of a distribution. The distribution and its indices are shown on the right. Each value in the distribution corresponds to one leaf of the decision tree (left).

with the backpropagation algorithm. The architecture of the network corresponds to that of a binary decision tree. Each node in this network represents a splitting and has therefore exactly two outputs where only one can be active at a time (Wang, Aggarwal, and Liu 2017).

The fourth and last category includes approaches that represent new layers in a neural network. Spatial transformation networks (Jaderberg et al. 2015) learn a transformation of the input tensor, which simplifies further processing in the network. In general this is a uniform representation of the input tensor which can be understood as spatial alignment. Since the accuracy of a mesh depends not only on the input, but also on the filters in convolution layers, (Jia et al. 2016) introduces a layer that learns to generate optimal filters based on the input. This layer consists of a small neural network with convolution and transposed convolution layers. A further possibility for the conditional adaptation of neural networks is the configuration of the weights over a temporal course as it was realized in (Holden, Komura, and Saito 2017) over a phase function. The authors of (Holden, Komura, and Saito 2017) used a Catmull-Rom spline as phase function which can also be replaced by a neural network. The additive component analysis (Murdock and De la Torre 2017) however tries to realize a non-linear dimension reduction by an approximation of additive functions. This is also defined as a fully connected layer and can be connected and trained in several layers. An approach based on this are the SplineNets (Keskin and Izadi 2018) which assign a new interpolated value to a learned spline via the response of a learned filter in the previous layer. This spline makes the function differentiable and several of these layers one behind the other can be understood as a topological graph.

Our novel approach is based on the idea of SplineNets (Keskin and Izadi 2018) to reduce convolutional complexity by simply mapping input characteristics to interpolated values. In addition, we simplify index generation with the general idea of binary neural networks (Courbariaux, Bengio, and David 2015). For this we use conditional distributions like random ferns (Bosch, Zisserman, and Munoz 2007). The indices are determined based on simple larger, smaller comparisons between input values. These indices are used to select weights from several distributions and multiply them by the input values. The indices itself are the evaluation of the decision tree and the selected weight is the leaf node. This means that we consider both the values in the distributions and the input and output values as probabilities. This allows us to train the whole new layer with the backpropagation algorithm together with the whole net, as well as to connect several layers in series. The reduction of the computation complexity comes like with SplineNets (Keskin and Izadi 2018) by the indexing which has to be calculated only once and not like with convolution layers, where a new convolution has to be calculated for each filter. In addition, our layer does not have to learn function parameters or perform expensive multiplications to generate the indices.

Due to the conditional weights which are trained holistically in one layer, our approach belongs to category 4. Since the indices generation is based on comparisons and random ferns (Bosch, Zisserman, and Munoz 2007) represent a concretisation of random forests (Breiman 2001), our approach also belongs to category 3. This means that it is a hybrid approach which is formalized as an independent layer but contains decision tree structures.

Our contributions in this work are:

1 A new layer that selects leaf weights based on binary decisions.

2 The approximation of filters for index generation by binary decisions.

3 A differentiable formal definition of the forward execution which is suitable for the backpropagation algorithm.

4 Analytical and empirical evaluation of the quality and runtime improvement compared to CNNs.

Methodology

The Figure 1 shows the core concept of our process. Random Ferns are binary decisions that are linked to conditional weights (see Figure 1). The binary decisions themselves represent the conditions. This means that it is a decision tree. Since each binary decision is always evaluated, the structure

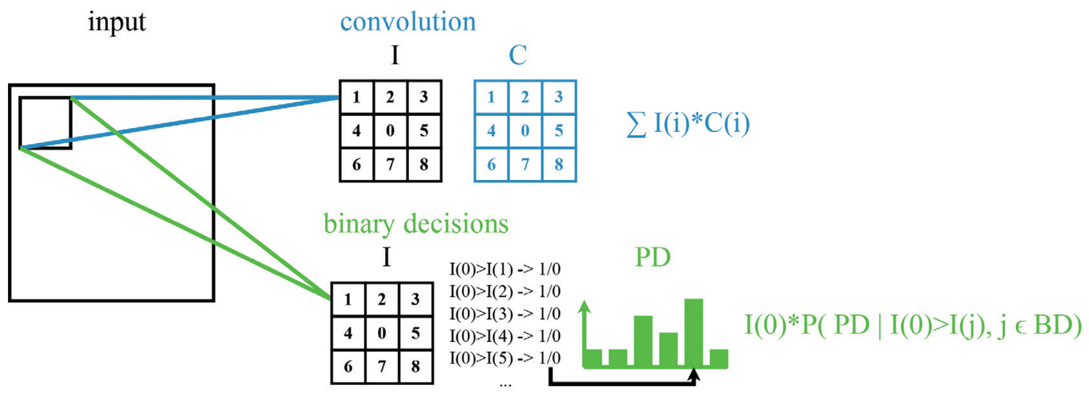

Figure 2: The functionality of the decision trees compared to a convolution. Like the convolution itself, an input window (I) is moved over the input tensor. In the upper part you can see the convolution which multiplies the weights (C) by the values in I and adds them up. In the lower part the decision tree is evaluated and the weight is multiplied by the central input value (0).

of this tree is arbitrary under the condition that each decision function must be contained once in each path, which makes the decision tree a balanced tree.

$$P(PD|I(k) > I(j), \forall j,k \in BD, j \neq k) \tag{1}$$

Equation 1 describes the evaluation of such a decision tree or Fern. I is the input tensor, PD the distribution (see Figure 1) and BD the indices of the comparisons. To use this decision tree now like a convolution the indices in BD refer only to values in an input window which is moved over the whole input tensor (see Figure 2). To combine several of these decision trees, the weights are multiplied. In the case of Equation 1, this would be the centered input values at the current window position making it easy to determine the derivative and thus the gradient. Another simplification of Equation 1 is to compare all positions in BD using only the central value (see Figure 2). This simplifies the back propagation of the error.

$$I(0) * P(PD|I(0) > I(j), \forall j \in BD) \tag{2}$$

This leads to Equation 2 which describes the evaluation of the decision tree for an input window. In the case of convolutions, this input window is not necessarily two dimensional, but also a tensor of weights. This tensor is represented by several distributions. Each depth value of the input tensor has its own distribution as with convolutions, where each depth uses its own two-dimensional weight matrix (see Figure 3).

This means that in the case of decision trees, each input depth has its own decision tree in the sense of its own distribution. For Equation 2 this means that each depth of the input tensor I_i with depth z indexes its own distribution PD_i over the same indexes BD.

$$\sum_{i=0}^{z-1} I_i(0) * P(PD_i|I_i(0) > I_i(j), \forall j \in BD) \tag{3}$$

Equation 3 describes the calculation where it has to be taken into account that each depth performs a multiplication with the central value and at the end, as with convolutions, the sum of all multiplications is computed. This summation makes it easier to determine the gradients for each distribution because there are no multiplicative dependencies between the distributions.

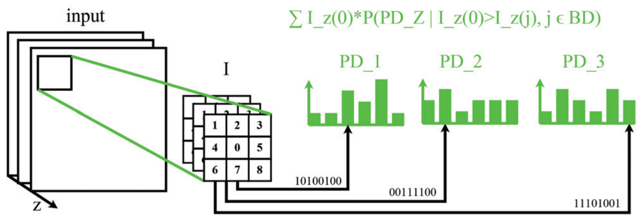

Figure 3: The use of multiple decision trees to process one input tensor. The indices of the comparisons (BD) are the same for all distributions (PD) which are different.

The next step describes the layer depth of the decision trees so that these decision trees can now also be used like convolution layers in neural networks (see Figure 4). As in the previous step, the same indexes BD are used for all layers but different distributions $PD_{l,i}$ are used for each layer. The reason for this is that the complexity of the calculation is reduced compared to convolutions.

Complexity: The calculation of a convolution layer with the input tensor t and n-times the convolution window c requires $t_x * t_y * t_z * n * c_x * c_y$ multiplications and additions. The decision trees, on the other hand, only have to determine the indices once, so that $n = 1$ can be set, thus reducing the complexity by the output depths. In addition, the multiplications are replaced by simple larger or smaller comparisons and a multiplication. From this it follows that $t_x * t_y * t_z * |BD|$ comparisons are performed and $t_x * t_y * t_z * n$ multiplications and additions.

$$O_l = \sum_{i=0}^{z-1} I_i(0) * P(PD_{l,i}|I_i(0) > I_i(j), \forall j \in BD) \tag{4}$$

To extend Equation 3 in this respect, each individual out-

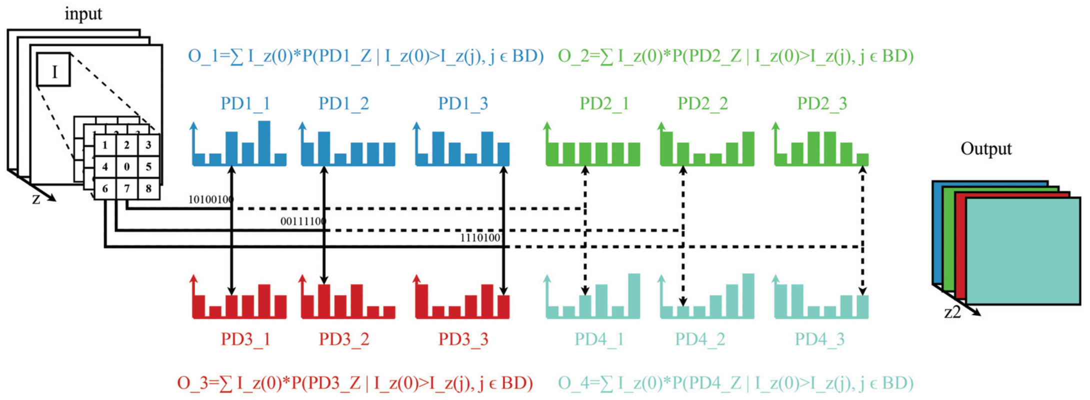
input
I
z
1 2 3
4 0 5
6 7 8
10100100
00111100
1110100
O_1=∑ I_z(0)*P(PD1_Z | I_z(0)>I_z(j), j ϵ BD)
O_2=∑ I_z(0)*P(PD2_Z | I_z(0)>I_z(j), j ϵ BD)
PD1_1
PD1_2
PD1_3
PD2_1
PD2_2
PD2_3
Output
z2
PD3_1
PD3_2
PD3_3
PD4_1
PD4_2
PD4_3
O_3=∑ I_z(0)*P(PD3_Z | I_z(0)>I_z(j), j ϵ BD)
O_4=∑ I_z(0)*P(PD4_Z | I_z(0)>I_z(j), j ϵ BD)

Figure 4: The use of different decision trees in multiple layers where the used indexes BD are the same. Each output layer is represented as its own color (red, blue, green, turquoise).

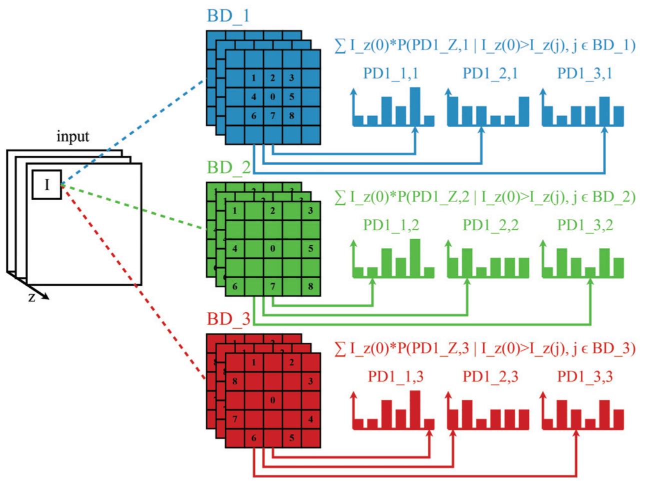
BD_1
∑ I_z(0)*P(PD1_Z,1 | I_z(0)>I_z(j), j ϵ BD_1)
PD1_1,1
PD1_2,1
PD1_3,1
input
I
z
BD_2
∑ I_z(0)*P(PD1_Z,2 | I_z(0)>I_z(j), j ϵ BD_2)
PD1_1,2
PD1_2,2
PD1_3,2
BD_3
∑ I_z(0)*P(PD1_Z,3 | I_z(0)>I_z(j), j ϵ BD_3)
PD1_1,3
PD1_2,3
PD1_3,3

Figure 5: Using the inception architecture for training the decision trees. The indexes BD_k are different for each color (red, green, blue) and are processed in parallel. Each of these index sets uses a distribution for each depth of the input tensor, as shown in Figure 3. The three index sets shown correspond to the output of one depth of the output layer. In the case of multiple output layers, the inception architecture is used as shown in Figure 4.

put layer O_l must be assigned a set of distributions $PD_{l,i}$. Equation 4 describes this change, but it is important to make sure that every tree uses the same BD indexes.

A disadvantage of the approach presented so far is that the size of the distributions grows exponentially $2^{|BD|}$. This means that the memory requirements can very quickly reach the limits of modern computers and the numerical calculation of very small numbers in large distributions can become too inaccurate. Another disadvantage of large distributions, i.e. a large number of binary comparisons, is that the probability that an index will be used during training decreases the larger the distribution is. For a convolution of the size 5×5 a distribution size of 2^{24} would be needed, which contains all comparisons with the central value. In order to make it possible to use several small distributions and still make it possible to cover larger input windows, we use the idea of inception architecture (Szegedy et al. 2015). This means that different index sets BD_k with depth b associated with different distributions $PD_{l,i,k}$ are aggregated in an output tensor. In our implementation we used the summation per layer.

$$O_l = \sum_{k=0}^{b-1} \sum_{i=0}^{z-1} I_i(0) * P(PD_{l,i,k} | I_i(0) > I_i(j), \quad \forall j \in BD_k) \tag{5}$$

Equation 5 describes the complete forward propagation per output layer of the presented new method for training decision trees in neural networks. All binary decision sets BD, with amount of sets b are used to compute the index for the assigned distributions $PD_{l,i,k}$. The sum of all selected weights in $PD_{l,i,k}$ multiplied with their corresponding input value $I_i(0)$ is calculated for each input window and written into the output tensor O_l. The bias term itself is omitted in the formulas to simplify them but is used as in conventional convolution layers.

The backward propagation of the error occurs inversely to the forward propagation. This means that as with convolution layers, a convolution with the error tensor takes place for each input value of the input tensor.

$$ERRI_i = \sum_{k=0}^{b-1} \sum_{l=0}^{L-1} ERRO_l(i) * P(PD_{l,i,k} | I_i(0) > I_i(j), \forall j \in BD_k) \tag{6}$$

Equation 6 describes the back propagation where L is the depth of the output tensor. Thus each value of the input layer i is assigned the sum of the errors $ERRO_l(i)$ multiplied by the indexed weights $PD_{l,i,k}$. In addition, for each value participated in the binary decisions the error is added divided by the size of the used binary decision set BD_k (Equation 7).

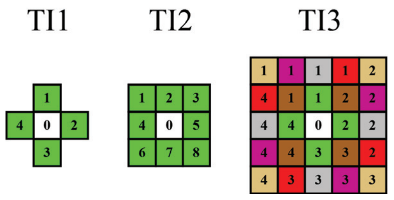

Figure 6: All index patterns (BD) used in our experiments. TI1 consists of four comparisons and the resulting distribution size of 2^4. TI2 is the full approximation of a 3×3 convolution with distribution size 2^8. TI3 is a pattern which uses the inception technique and consists of six distributions. Each pattern in TI3 has the same color and consists therefore out of four indexes. The full parameter size for one TI3 pattern is $6 * (2^4)$.

$$ERRI_i(j,k) += \sum_{l=0}^{L-1} ERRO_l(i) * P(PD_{l,i,k} | I_i(0) > I_i(j), \forall j \in BD_k)/|BD_k| \quad (7)$$

Equation 7 is calculated for each index j in each used binary decision set BD_k and sums the error over the output tensor of the depth L. The division by the record size $|BD_k|$ results in an equal share of the error being assigned to each index. This is due to the fact that the participation in the resulting error is independent of the binary value of the evaluation from the decision function.

$$Grad(PD_{l,i,k}) = \sum_{l=0}^{L-1} I_i(0) * ERRO_l(i) \quad (8)$$

To determine the gradient, only the derivation between the generated error and the input needs to be considered. This is described in Equation 8 and shows that only the central value of the input window and the output value are required. For the binary decision functions, the derivation is 0, since these are independent of the weights in the distribution.

Experiments

Figure 6 shows the used index patterns for our evaluations. We used the models LeNet-5 (LeCun et al. 1998) with rectifier linear units (ReLu) instead of the hyperbolic tangent function and the deep residual model with depth 16 (ResNet-16) and 34 (ResNet-34). In both models (ResNet-16 and ResNet-34) we used a batch normalization block after each convolution or decision tree layer. The LeNet-5 model was used in the comparison on the MNIST (LeCun et al. 1998) dataset with the index patterns TI2 and TI3 (Figure 6). The ResNet-34 was used for the comparison on the CIFAR10 (Krizhevsky and Hinton 2009) dataset with the TI1 pattern. As an alternative evaluation for image classification we used landmark regression. Therefore, we compared the decision trees with convolutions on the 300W (Zhu and Ramanan 2012) dataset using the ResNet-16 and the TI1 patterns.

The general idea behind our experiments is not to surpass the state-of-the-art, but to compare decision trees with convolutions in the same architecture. For this purpose we tried to get as close as possible to the results of the state-of-the-art with simple means and to design the training process for convolutions and decision trees in the same way.

MNIST consists of 70,000 hand written digits and has therefore ten classes. Each image has a resolution of 28×28 pixels and is provided as gray scale image. The training set contains 60,000 and the test set 10,000 images. This data set is size-normalized and centered and represents a subset of the larger NIST dataset. As evaluation metric the classification accuracy ($\frac{Correct\ Classification}{Test\ set\ size}$) is used. We only report the best result as it is done for the state-of-the-art (Wan et al. 2013; Cireşan, Meier, and Schmidhuber 2012; Sato, Nishimura, and Yokoi 2015).

CIFAR10 consists of 60,000 color images with ten different categories. Each image has a resolution of 32×32 pixels and is provided in the RGB format. The training set contains 50,000 and the test set 10,000 images. As evaluation metric the classification accuracy ($\frac{Correct\ Classification}{Test\ set\ size}$) is used. We only report the best result as it is done for the state-of-the-art (Graham 2014; Springenberg et al. 2014; Mishkin and Matas 2015).

300W is an aggregation of multiple datasets (LFPW (Belhumeur et al. 2013), HELEN (Le et al. 2012), AFW (Zhu and Ramanan 2012) and XM2VTS (Messer et al. 1999)). The training set consists of 3,148 face images from the LFPW and HELEN dataset. For the test set 689 images are provided. Each image has 68 annotated landmarks (Sagonas et al. 2013). In the evaluation the test set is separated into three categories the full set, the challenging set (iBUG, 135 images) and the common set (LFPW and HELEN, 554 images). As evaluation metric we used the normalized mean errors (NME) which corresponds to the average distance between detected and annotated landmark, normalized by the pixel distance between both eye centers. This is the same evaluation procedure as the state-of-the-art (Feng et al. 2018; Dong et al. 2018; Ren et al. 2016).

Training parameters for MNIST: We used the Adam optimizer (Kingma and Ba 2014) with the first momentum set to 0.9 and the second momentum set to 0.999. Weight decay was set to $5 * 10^{-4}$ for the convolutions and to 10^{-8} for the decision trees. The batch size was set to 400 and each batch was always balanced in terms of available classes. This means that in each batch each class was represented 40 times. The initial learning rate was set to 10^{-2} and reduced by 10^{-1} after each 100 epochs until it reached 10^{-4}. For the learning rate of 10^{-4} we continued the training for additional 1000 epochs and selected the best result. For data augmentation we used random noise in the range of 0-30% of the image resolution.

Training parameters for CIFAR10: We used the Adam optimizer (Kingma and Ba 2014) with the first momentum set to 0.9 and the second momentum set to 0.999. Weight decay was set to $5 * 10^{-5}$ for the convolutions and to 10^{-10} for the decision trees. The batch size was set to 50 with the same batch balancing approach as for the MNIST dataset. For CIFAR this means each batch consisted of five examples per class. The initial learning rate was set to 10^{-2} and reduced by 10^{-1} after each 500 epochs until it reached 10^{-5}. For the learning rate of 10^{-5} we continued the training for additional 1000 epochs and selected the best result. For data augmentation we used random cropping of 24×24 patches, random color offsets, random color distortion, flipping the image horizontally and vertically as well as random noise in the range of 0-20% of the image resolution. Additionally, we overlayed patches of the same class with an intensity of up to 20%.

Training parameters for 300W: All images where resized to 80×80 pixels. We used the Adam optimizer (Kingma and Ba 2014) with the first momentum set to 0.9 and the second momentum set to 0.999. Weight decay was set to $5 * 10^{-4}$ for the convolutions and to 10^{-8} for the decision trees. The batch size was set to 30. All 20 iterations an evaluation of the landmark accuracy was performed for the test and the training set. The accuracy on the training set was used to balance the batches. This was done by splitting the training set into three categories. The first category are the most inaccurate 20%. For the second category we used the range between the first category and the most inaccurate 50%. The last category is the range between the second category and the most inaccurate 80%. For each batch we selected ten examples out of each category. The initial learning rate was set to 10^{-6} and increased by 10^{1} after each 100 epochs until it reached 10^{-4}. For the learning rate of 10^{-4} we continued the training for additional 1000 epochs. Afterwards, we reduced the learning rate after each 100 epochs by 10^{-1} until we reached 10^{-7} and stopped the training. For data augmentation we used random noise in the range of 0-20% of the image resolution. The image and landmarks where randomly shifted by up to 20% of the image resolution into each direction. Additionally, we added randomly Gaussian blur ($\sigma = [1.0, 1.3]$). For occlusions we overlayed up to three boxes and filled them either with a fixed random value or a random value for each pixel in the box. We also randomly changed the contrast of the image in the range [-40, 40].

Hardware and implementation: For training and evaluation we used two different hardware setups. For LeNet-5 we used a desktop PC with an Intel i5-4570 CPU (3.2 GHz), 16 GB DDR4 RAM, NVIDIA GTX 1050Ti GPU with 4GB RAM and Windows 7 64 bit operating system. The second hardware setup was used for the ResNet models since those require more GPU RAM. Therefore, we used a server with an Intel i9-9900K CPU (3.6 GHz), 64 GB DDR4 RAM, two RTX 2080ti GPUs with 11.2GB RAM each and an Windows 8.1 64 bit operating system. We implemented the decision tree layer in C++ on the CPU and in CUDA on the GPU. The implementation was integrated into the DLIB (King 2009) framework which uses CUDNN functions. An implementation for Tensorflow (Abadi et al. 2016) and Torch (Collobert, Bengio, and Mariéthoz 2002) is also planned since those are currently the most popular frameworks.

Table 1: Comparison between the proposed decision tree layers and convolutions on the MNIST dataset in terms of classification accuracy. As index patterns for the decision tree we used TI2 and TI3. The current sate-of-the-art is shown in gray.

Method	Result
LeNet-5(TI2)	99.23
LeNet-5(TI3)	99.48
LeNet-5(Conv.)	99.37
(Wan et al. 2013)	99.79
(Cireşan, Meier, and Schmidhuber 2012)	99.77
(Sato, Nishimura, and Yokoi 2015)	99.77

Table 1 shows the results of our adapted Le-Net5 model. As can be seen the TI2 and TI3 pattern (Figure 6) perform similar to the convolutions. The TI2 pattern is an approximation of a 3×3 convolution and achieves a classification accuracy of 99.23%. In comparison to this the 5×5 convolutions as used in the LeNet-5 model achieve an accuracy of 99.37% which is an improvement of 0.14%. Approximating the 5×5 convolutions with the TI3 pattern and the inception technique achieves 99.48. If the runtime is also considered (Table 4), it can be seen that the use of the decision trees requires only one third of the computing time in comparison to the convolutions (evaluation on only one CPU core). A disadvantage of the decision trees, on the other hand, is the increased memory consumption. In the case of the LeNet-5 model, both convolution layers require $((1*5*5*6)+6)+((6*5*5*16)+16) = 2572$ parameter ($Input * Conv * Output + Bias$). The TI2 pattern needs $((1*1*256*6)+6)+((6*1*256*16)+16) = 26134$ parameters and the TI3 pattern needs $((1*6*16*6)+6)+((6*6*16*16)+16) = 9814$ ($Input * Inceptionwidth * Distributionsize * Output + Bias$).

Table 2: Comparison between the proposed decision tree layers and convolutions on the CIFAR10 dataset in terms of classification accuracy. As index patterns for the decision tree we used TI1. The current sate-of-the-art is shown in gray.

Method	Result
ResNet-34(TI1)	92.20
ResNet-34(Conv.)	91.12
(Graham 2014)	96.53
(Springenberg et al. 2014)	95.59
(Mishkin and Matas 2015)	94.16

Table 2 shows the comparison between the TI1 pattern and the convolutions using the ResNet-34 model. As can be seen both achieve classification accuracies above 90%. The TI1 pattern performs slightly better in comparison to the convolutions (1.08 improvement). Comparing the runtime (Table 4) of both approaches it can be seen that the decision trees are significantly faster to compute (6.77ms vs 18.1ms). The memory consumption for one distribution of the TI1 pattern is 16 floats, for one convolution 9 floats (3×3). This means that the parameters of the model are almost doubled while the runtime is only one third.

Table 3: Comparison between the proposed decision tree layers and convolutions on the CIFAR10 dataset in terms of average normalized mean error on the 300W dataset. As index patterns for the decision tree we used TI1. The current sate-of-the-art is shown in gray.

Method	Comm.	Challe.	Full
ResNet-16(TI1)	3.64	6.81	4.26
ResNet-16(Conv.)	3.69	6.76	4.29
SAN_{GT} (Dong et al. 2018)	3.34	6.60	3.98
SAN_{OD} (Dong et al. 2018)	3.41	7.55	4.24
ResNet-50 (Feng et al. 2018)	3.01	6.01	3.60
CNN-6/7 (Feng et al. 2018)	3.27	7.18	4.10
LAB (8-stack) (Wu et al. 2018)	3.42	6.98	4.12
3DDFA (Zhu et al. 2016)	6.15	10.59	7.01
LBF (Ren et al. 2016)	4.95	11.98	6.32
SDM (Xiong and De la Torre 2013)	5.60	15.40	7.52

Table 3 shows the results for landmark regression using the ResNet-16 model. As can be seen the convolutions and the decision trees achieve nearly the same result. We used the same pattern (TI1) as for the CIFAR10 classification which means that the memory consumption of the decision trees is nearly twice as high as for the convolutions. The runtime in contrast is halved (4,64ms vs 10.76ms).

Table 4: Runtime comparison for different models and index patterns in milliseconds. For the convolutions we used the CUDNN implementation with the best selected algorithm.

Model	Input	HW	Runtime
LeNet-5 (TI2)	$28 \times 28 \times 1$	1 CPU Core	0.18ms
LeNet-5 (TI3)	$28 \times 28 \times 1$	1 CPU Core	0.36ms
LeNet-5 (Conv.)	$28 \times 28 \times 1$	1 CPU Core	0.83ms
ResNet-34 (TI1)	$32 \times 32 \times 3$	GPU 1050ti	6.77ms
ResNet-34 (Conv.)	$32 \times 32 \times 3$	GPU 1050ti	18.1ms
ResNet-16 (TI1)	$80 \times 80 \times 3$	GPU 1050ti	4.64ms
ResNet-16 (Conv.)	$80 \times 80 \times 3$	GPU 1050ti	10.76ms

Table 4 shows an overview of all runtimes of the models used using convolution and decision trees. All runtime evaluations were performed on a single CPU core (Intel i5-4570) or an NVIDIA 1050ti GPU to ensure reproducibility and to simplify the comparison to other hardware environments.

Conclusions and Discussions

We presented a novel approach for training decision trees in neural network architectures using the back propagation algorithm and showed that it is possible to achieve the same results as with convolutions. Classification and a regression experiment where conducted on publicly available datasets. The improved runtime of the decision trees was estimated theoretically and empirically shown for different models against the high performance CUDNN implementation from NVIDIA. From an industrial point of view, reducing the runtime while maintaining or even improving the predictive quality is a desirable improvement. In contrast to the runtime, the increased memory consumption is a disadvantage. Further research should investigate the use of indexing sets with different depths and the reduction of the decision trees to only necessary paths. Here the authors see further opportunities for the reduction of the computation time and the memory consumption. In addition, the decision trees could also be extended to only use binary weights as it is done in binary convolution neuronal networks (Courbariaux, Bengio, and David 2015). This would reduce the runtime and memory consumption.

Acknowledgments

Work of the authors is supported by the Institutional Strategy of the University of Tübingen (Deutsche Forschungsgemeinschaft, ZUK 63).

References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 265–283.

Baek, S.; Kim, K. I.; and Kim, T.-K. 2017. Deep convolutional decision jungle for image classification. *arXiv preprint arXiv:1706.02003*.

Belhumeur, P. N.; Jacobs, D. W.; Kriegman, D. J.; and Kumar, N. 2013. Localizing parts of faces using a consensus of exemplars. *Pattern Analysis and Machine Intelligence* 35(12):2930–2940.

Bicici, U. C.; Keskin, C.; and Akarun, L. 2018. Conditional information gain networks. In *International Conference on Pattern Recognition*, 1390–1395. IEEE.

Bosch, A.; Zisserman, A.; and Munoz, X. 2007. Image classification using random forests and ferns. In *International Conference on Computer Vision*, 1–8. IEEE.

Breiman, L. 2001. Random forests. *Machine learning* 45(1):5–32.

Chen, T. Q.; Rubanova, Y.; Bettencourt, J.; and Duvenaud, D. K. 2018. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, 6571–6583.

Cireşan, D.; Meier, U.; and Schmidhuber, J. 2012. Multi-column deep neural networks for image classification. *arXiv preprint arXiv:1202.2745*.

Collobert, R.; Bengio, S.; and Mariéthoz, J. 2002. Torch: a modular machine learning software library. Technical report, Idiap.

Courbariaux, M.; Bengio, Y.; and David, J.-P. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, 3123–3131. IEEE.

Denoyer, L., and Gallinari, P. 2014. Deep sequential neural network. *arXiv preprint arXiv:1410.0510*.

Dong, X.; Yan, Y.; Ouyang, W.; and Yang, Y. 2018. Style aggregated network for facial landmark detection. In *Conference on Computer Vision and Pattern Recognition*, volume 2, 6. IEEE.

Feng, Z.; Kittler, J.; Awais, M.; Huber, P.; and Wu, X. 2018. Wing loss for robust facial landmark localisation with convolutional neural networks. In *Conference on Computer Vision and Pattern Recognition*, volume 1, 3. IEEE.

Graham, B. 2014. Fractional max-pooling. *CoRR* abs/1412.6071.

Holden, D.; Komura, T.; and Saito, J. 2017. Phase-functioned neural networks for character control. *Transactions on Graphics* 36(4):42.

Ioannou, Y.; Robertson, D.; Zikic, D.; Kontschieder, P.; Shotton, J.; Brown, M.; and Criminisi, A. 2016. Decision forests, convolutional networks and the models in-between. *arXiv preprint arXiv:1603.01250*.

Jaderberg, M.; Simonyan, K.; Zisserman, A.; et al. 2015. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, 2017–2025.

Jia, X.; De Brabandere, B.; Tuytelaars, T.; and Gool, L. V. 2016. Dynamic filter networks. In *Advances in Neural Information Processing Systems*, 667–675.

Keskin, C., and Izadi, S. 2018. Splinenets: Continuous neural decision graphs. In *Advances in Neural Information Processing Systems*, 1994–2004.

King, D. E. 2009. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research* 10(Jul):1755–1758.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kontschieder, P.; Fiterau, M.; Criminisi, A.; and Rota Bulo, S. 2015. Deep neural decision forests. In *International Conference on Computer Vision*, 1467–1475. IEEE.

Krizhevsky, A., and Hinton, G. 2009. Learning multiple layers of features from tiny images. Technical report, Citeseer.

Le, V.; Brandt, J.; Lin, Z.; Bourdev, L.; and Huang, T. S. 2012. Interactive facial feature localization. In *European Conference on Computer Vision*, 679–692. Springer.

LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P.; et al. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.

Messer, K.; Matas, J.; Kittler, J.; Luettin, J.; and Maitre, G. 1999. Xm2vtsdb: The extended m2vts database. In *Audio and Video-based Biometric Person Authentication*, volume 964, 965–966.

Mishkin, D., and Matas, J. 2015. All you need is a good init. *arXiv preprint arXiv:1511.06422*.

Murdock, C., and De la Torre, F. 2017. Additive component analysis. In *Conference on Computer Vision and Pattern Recognition*, 2491–2499. IEEE.

Ren, S.; Cao, X.; Wei, Y.; and Sun, J. 2016. Face alignment via regressing local binary features. *Image Processing* 25(3):1233–1245.

Rota Bulo, S., and Kontschieder, P. 2014. Neural decision forests for semantic image labelling. In *Computer Vision and Pattern Recognition*, 81–88. IEEE.

Sagonas, C.; Tzimiropoulos, G.; Zafeiriou, S.; and Pantic, M. 2013. A semi-automatic methodology for facial landmark annotation. In *Conference on Computer Vision and Pattern Recognition Workshops*, 896–903. IEEE.

Sato, I.; Nishimura, H.; and Yokoi, K. 2015. Apac: Augmented pattern classification with neural networks. *arXiv preprint arXiv:1505.03229*.

Shazeer, N.; Mirhoseini, A.; Maziarz, K.; Davis, A.; Le, Q. V.; Hinton, G. E.; and Dean, J. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *CoRR* abs/1701.06538.

Springenberg, J. T.; Dosovitskiy, A.; Brox, T.; and Riedmiller, M. 2014. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *Computer Vision and Pattern Recognition*, 1–9. IEEE.

Wan, L.; Zeiler, M.; Zhang, S.; Le Cun, Y.; and Fergus, R. 2013. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, 1058–1066. IEEE.

Wang, S.; Aggarwal, C.; and Liu, H. 2017. Using a random forest to inspire a neural network and improving on it. In *International Conference on Data Mining*, 1–9. SIAM.

Wu, W.; Qian, C.; Yang, S.; Wang, Q.; Cai, Y.; and Zhou, Q. 2018. Look at boundary: A boundary-aware face alignment algorithm. In *Computer Vision and Pattern Recognition*. IEEE.

Xiong, X., and De la Torre, F. 2013. Supervised descent method and its applications to face alignment. In *Computer Vision and Pattern Recognition*, 532–539. IEEE.

Xiong, C.; Zhao, X.; Tang, D.; Jayashree, K.; Yan, S.; and Kim, T.-K. 2015. Conditional convolutional neural network for modality-aware face recognition. In *International Conference on Computer Vision*, 3667–3675.

Zhu, X., and Ramanan, D. 2012. Face detection, pose estimation, and landmark localization in the wild. In *Computer Vision and Pattern Recognition*, 2879–2886. IEEE.

Zhu, X.; Lei, Z.; Liu, X.; Shi, H.; and Li, S. Z. 2016. Face alignment across large poses: A 3d solution. In *Conference on Computer Vision and Pattern Recognition*, 146–155. IEEE.