Deep Neural Decision Forests

Peter Kontschieder[1] Madalina Fiterau[*,2] Antonio Criminisi[1] Samuel Rota Bulò[1,3]

Microsoft Research[1] Cambridge, UK
Carnegie Mellon University[2] Pittsburgh, PA
Fondazione Bruno Kessler[3] Trento, Italy

Abstract

We present Deep Neural Decision Forests – a novel approach that unifies classification trees with the representation learning functionality known from deep convolutional networks, by training them in an end-to-end manner. To combine these two worlds, we introduce a stochastic and differentiable decision tree model, which steers the representation learning usually conducted in the initial layers of a (deep) convolutional network. Our model differs from conventional deep networks because a decision forest provides the final predictions and it differs from conventional decision forests since we propose a principled, joint and global optimization of split and leaf node parameters. We show experimental results on benchmark machine learning datasets like MNIST and ImageNet and find on-par or superior results when compared to state-of-the-art deep models. Most remarkably, we obtain Top5-Errors of only $7.84\%/6.38\%$ *on ImageNet validation data when integrating our forests in a single-crop, single/seven model GoogLeNet architecture, respectively. Thus, even without any form of training data set augmentation we are improving on the 6.67% error obtained by the best GoogLeNet architecture (7 models, 144 crops).*

1. Introduction

Random forests [1, 4, 7] have a rich and successful history in machine learning in general and the computer vision community in particular. Their performance has been empirically demonstrated to outperform most state-of-the-art learners when it comes to handling high dimensional data problems [6], they are inherently able to deal with multi-class problems, are easily distributable on parallel hardware architectures while being considered to be close to an ideal learner [11]. These facts and many (computationally) appealing properties make them attractive for various research areas and commercial products. In such a way, random forests could be used as out-of-the-box classifiers for many computer vision tasks such as image classification [3] or semantic segmentation [5, 32], where the input space (and corresponding data representation) they operate on is typically predefined and left unchanged.

One of the consolidated findings of modern, (very) deep learning approaches [19, 23, 36] is that their joint and unified way of learning feature representations together with their classifiers greatly outperforms conventional feature descriptor & classifier pipelines, whenever enough training data and computation capabilities are available. In fact, the recent work in [12] demonstrated that deep networks could even outperform humans on the task of image classification. Similarly, the success of deep networks extends to speech recognition [38] and automated generation of natural language descriptions of images [9].

Addressing random forests to learn both, proper representations of the input data and the final classifiers in a joint manner is an open research field that has received little attention in the literature so far. Notable but limited exceptions are [18, 24] where random forests were trained in an entangled setting, stacking intermediate classifier outputs with the original input data. The approach in [28] introduced a way to integrate multi-layer perceptrons as split functions, however, representations were learned only locally at split node level and independently among split nodes. While these attempts can be considered early forms of representation learning in random forests, their prediction accuracies remained below the state-of-the-art.

In this work we present *Deep Neural Decision Forests* – a novel approach to unify appealing properties from representation learning as known from deep architectures with the divide-and-conquer principle of decision trees. We introduce a stochastic, differentiable, and therefore back-propagation compatible version of decision trees, guiding the representation learning in lower layers of deep convolutional networks. Thus, the task for representation learning is to reduce the uncertainty on the routing decisions of a sample taken at the split nodes, such that a globally defined

[*]The major part of this research project was undertaken when Madalina was an intern with Microsoft Research Cambridge, UK.

loss function is minimized.

Additionally, the optimal predictions for all leaves of our trees given the split decisions can be obtained by minimizing a convex objective and we provide an optimization algorithm for it that does not resort to tedious step-size selection. Therefore, at test time we can take the optimal decision for a sample ending up in the leaves, with respect to all the training data and the current state of the network.

Our realization of back-propagation trees is modular and we discuss how to integrate them in existing deep learning frameworks such as Caffe [16], MatConvNet [37], Minerva[1], *etc*. supported by standard neural network layer implementations. Of course, we also maintain the ability to use back-propagation trees as (shallow) stand-alone classifiers. We demonstrate the efficacy of our approach on a range of datasets, including MNIST and ImageNet, showing superior or on-par performance with the state-of-the-art.

Related Works. The main contribution of our work relates to enriching decision trees with the capability of representation learning, which requires a tree training approach departing from the prevailing greedy, local optimization procedures typically employed in the literature [7]. To this end, we will present the parameter learning task in the context of empirical risk minimization. Related approaches of tree training via global loss function minimization were *e.g*. introduced in [30] where during training a globally tracked weight distribution guides the optimization, akin to concepts used in boosting. The work in [15] introduced regression tree fields for the task of image restoration, where leaf parameters were learned to parametrize Gaussian conditional random fields, providing different types of interaction. In [35], fuzzy decision trees were presented, including a training mechanism similar to back-propagation in neural networks. Despite sharing some properties in the way parent-child relationships are modeled, our work differs as follows: i) We provide a globally optimal strategy to estimate predictions taken in the leaves (whereas [35] simply uses histograms for probability mass estimation). ii) The aspect of representation learning is absent in [35] and iii) We do not need to specify additional hyper-parameters which they used for their routing functions (which would potentially account for millions of additional hyper-parameters needed in the ImageNet experiments). The work in [24] investigated the use of sigmoidal functions for the task of differentiable information gain maximization. In [25], an approach for global tree refinement was presented, proposing joint optimization of leaf node parameters for trained trees together with pruning strategies to counteract overfitting. The work in [26] describes how (greedily) trained, cascaded random forests can be represented by deep networks (and refined by additional training), building upon the work in [31] (which describes the mapping of decision trees into multi-layer neural networks).

In [2], a Bayesian approach using priors over all parameters is introduced, where also sigmoidal functions are used to model splits, based on linear functions on the input (*c.f*. the non-Bayesian work from Jordan [17]). Other hierarchical mixture of expert approaches can also be considered as tree-structured models, however, lacking both, representation learning and ensemble aspects.

[1] https://github.com/dmlc/minerva

2. Decision Trees with Stochastic Routing

Consider a classification problem with input and (finite) output spaces given by $\mathcal{X}$ and $\mathcal{Y}$, respectively. A *decision tree* is a tree-structured classifier consisting of decision (or split) nodes and prediction (or leaf) nodes. Decision nodes indexed by $\mathcal{N}$ are internal nodes of the tree, while prediction nodes indexed by $\mathcal{L}$ are the terminal nodes of the tree. Each prediction node $\ell \in \mathcal{L}$ holds a probability distribution $\boldsymbol{\pi}_\ell$ over $\mathcal{Y}$. Each decision node $n \in \mathcal{N}$ is instead assigned a decision function $d_n(\cdot;\Theta) : \mathcal{X} \rightarrow [0,1]$ parametrized by Θ, which is responsible for routing samples along the tree. When a sample $\boldsymbol{x} \in \mathcal{X}$ reaches a decision node n it will be sent to the left or right subtree based on the output of $d_n(\boldsymbol{x};\Theta)$. In standard decision forests, d_n is binary and the routing is deterministic. In this paper we will consider rather a probabilistic routing, *i.e*. the routing direction is the output of a Bernoulli random variable with mean $d_n(\boldsymbol{x};\Theta)$. Once a sample ends in a leaf node ℓ, the related tree prediction is given by the class-label distribution $\boldsymbol{\pi}_\ell$. In the case of stochastic routings, the leaf predictions will be averaged by the probability of reaching the leaf. Accordingly, the final prediction for sample $\boldsymbol{x}$ from tree T with decision nodes parametrized by Θ is given by

$$\mathbb{P}_T[y|\boldsymbol{x},\Theta,\boldsymbol{\pi}] = \sum_{\ell\in\mathcal{L}} \pi_{\ell y}\mu_\ell(\boldsymbol{x}|\Theta)\,, \qquad (1)$$

where $\boldsymbol{\pi} = (\boldsymbol{\pi}_\ell)_{\ell\in\mathcal{L}}$ and $\pi_{\ell y}$ denotes the probability of a sample reaching leaf ℓ to take on class y, while $\mu_\ell(\boldsymbol{x}|\Theta)$ is regarded as the *routing function* providing the probability that sample $\boldsymbol{x}$ will reach leaf ℓ. Clearly, $\sum_\ell \mu_\ell(\boldsymbol{x}|\Theta) = 1$ for all $\boldsymbol{x} \in \mathcal{X}$.

In order to provide an explicit form for the routing function we introduce the following binary relations that depend on the tree's structure: $\ell \swarrow n$, which is true if ℓ belongs to the left subtree of node n, and $n \searrow \ell$, which is true if ℓ belongs to the right subtree of node n. We can now exploit these relations to express μ_ℓ as follows:

$$\mu_\ell(\boldsymbol{x}|\Theta) = \prod_{n\in\mathcal{N}} d_n(\boldsymbol{x};\Theta)^{\mathbb{1}_{\ell\swarrow n}} \bar{d}_n(\boldsymbol{x};\Theta)^{\mathbb{1}_{n\searrow\ell}}\,, \qquad (2)$$

where $\bar{d}_n(\boldsymbol{x};\Theta) = 1 - d_n(\boldsymbol{x};\Theta)$, and $\mathbb{1}_P$ is an indicator function conditioned on the argument P. Although the

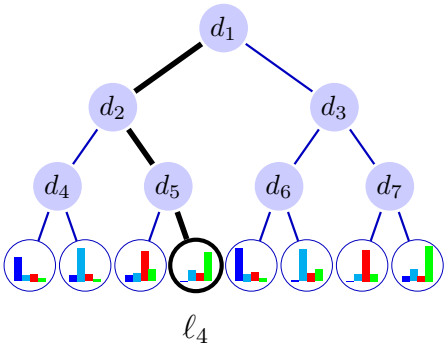

Figure 1. Each node $n \in \mathcal{N}$ of the tree performs routing decisions via function $d_n(\cdot)$ (we omit the parametrization Θ). The black path shows an exemplary routing of a sample $\boldsymbol{x}$ along a tree to reach leaf ℓ_4, which has probability $\mu_{\ell_4} = d_1(\boldsymbol{x})\bar{d}_2(\boldsymbol{x})\bar{d}_5(\boldsymbol{x})$.

product in (2) runs over all nodes, only decision nodes along the path from the root node to the leaf ℓ contribute to μ_ℓ, because for all other nodes $\mathbb{1}_{\ell \swarrow n}$ and $\mathbb{1}_{n \searrow \ell}$ will be both 0 (assuming $0^0 = 1$, see Fig. 1 for an illustration).

Decision nodes In the rest of the paper we consider decision functions delivering a stochastic routing with decision functions defined as follows:

$$d_n(\boldsymbol{x}; \Theta) = \sigma(f_n(\boldsymbol{x}; \Theta)) \,, \tag{3}$$

where $\sigma(x) = (1 + e^{-x})^{-1}$ is the sigmoid function, and $f_n(\cdot; \Theta) : \mathcal{X} \to \mathbb{R}$ is a real-valued function depending on the sample and the parametrization Θ. Further details about the functions f_n can be found in Section 4.1, but intuitively depending on how we choose these functions we can model trees having shallow decisions (*e.g.* such as in oblique forests [13]) as well as deep ones.

Forests of decision trees. A forest is an ensemble of decision trees $\mathcal{F} = \{T_1, \dots, T_{\mathsf{k}}\}$, which delivers a prediction for a sample $\boldsymbol{x}$ by averaging the output of each tree, *i.e.*

$$\mathbb{P}_{\mathcal{F}}[y|\boldsymbol{x}] = \frac{1}{\mathsf{k}} \sum_{h=1}^{\mathsf{k}} \mathbb{P}_{T_h}[y|\boldsymbol{x}] \,, \tag{4}$$

omitting the tree parameters for notational convenience.

3. Learning Trees by Back-Propagation

Learning a decision tree modeled as in Section 2 requires estimating both, the decision node parametrizations Θ and the leaf predictions $\boldsymbol{\pi}$. For their estimation we adhere to the minimum empirical risk principle with respect to a given data set $\mathcal{T} \subset \mathcal{X} \times \mathcal{Y}$ under log-loss, *i.e.* we search for the minimizers of the following risk term:

$$R(\Theta, \boldsymbol{\pi}; \mathcal{T}) = \frac{1}{|\mathcal{T}|} \sum_{(\boldsymbol{x},y) \in \mathcal{T}} L(\Theta, \boldsymbol{\pi}; \boldsymbol{x}, y) \,, \tag{5}$$

where $L(\Theta, \boldsymbol{\pi}; \boldsymbol{x}, y)$ is the log-loss term for the training sample $(\boldsymbol{x}, y) \in \mathcal{T}$, which is given by

$$L(\Theta, \boldsymbol{\pi}; \boldsymbol{x}, y) = -\log(\mathbb{P}_T[y|\boldsymbol{x}, \Theta, \boldsymbol{\pi}]) \,, \tag{6}$$

and $\mathbb{P}_T$ is defined as in (1).

We consider a two-step optimization strategy, described in the rest of this section, where we alternate updates of Θ with updates of $\boldsymbol{\pi}$ in a way to minimize (5).

3.1. Learning Decision Nodes

All decision functions depend on a common parameter Θ, which in turn parametrizes each function f_n in (3). So far, we made no assumptions about the type of functions in f_n, therefore nothing prevents the optimization of the risk with respect to Θ for a given $\boldsymbol{\pi}$ from eventually becoming a difficult and large-scale optimization problem. As an example, Θ could absorb all the parameters of a deep neural network having f_n as one of its output units. For this reason, we will employ a Stochastic Gradient Descent (SGD) approach to minimize the risk with respect to Θ, as commonly done in the context of deep neural networks:

$$\begin{aligned} \Theta^{(t+1)} &= \Theta^{(t)} - \eta \frac{\partial R}{\partial \Theta}(\Theta^{(t)}, \boldsymbol{\pi}; \mathcal{B}) \\ &= \Theta^{(t)} - \frac{\eta}{|\mathcal{B}|} \sum_{(\boldsymbol{x},y) \in \mathcal{B}} \frac{\partial L}{\partial \Theta}(\Theta^{(t)}, \boldsymbol{\pi}; \boldsymbol{x}, y) \end{aligned} \tag{7}$$

Here, $0 < \eta$ is the learning rate and $\mathcal{B} \subseteq \mathcal{T}$ is a random subset (*a.k.a.* mini-batch) of samples from the training set. Although not shown explicitly, we additionally consider a momentum term to smooth out the variations of the gradients. The gradient of the loss L with respect to Θ can be decomposed by the chain rule as follows

$$\frac{\partial L}{\partial \Theta}(\Theta, \boldsymbol{\pi}; \boldsymbol{x}, y) = \sum_{n \in \mathcal{N}} \frac{\partial L(\Theta, \boldsymbol{\pi}; \boldsymbol{x}, y)}{\partial f_n(\boldsymbol{x}; \Theta)} \frac{\partial f_n(\boldsymbol{x}; \Theta)}{\partial \Theta} \,. \tag{8}$$

Here, the gradient term that depends on the decision tree is given by

$$\frac{\partial L(\Theta, \boldsymbol{\pi}; \boldsymbol{x}, y)}{\partial f_n(\boldsymbol{x}; \Theta)} = d_n(\boldsymbol{x}; \Theta) A_{n_r} - \bar{d}_n(\boldsymbol{x}; \Theta) A_{n_l} \,, \tag{9}$$

where n_l and n_r indicate the left and right child of node n, respectively, and we define A_m for a generic node $m \in \mathcal{N}$ as

$$A_m = \frac{\sum_{\ell \in \mathcal{L}_m} \pi_{\ell y} \mu_\ell(\boldsymbol{x}|\Theta)}{\mathbb{P}_T[y|\boldsymbol{x}, \Theta, \boldsymbol{\pi}]} \,.$$

With $\mathcal{L}_m \subseteq \mathcal{L}$ we denote the set of leaves held by the subtree rooted in node m. Detailed derivations of (9) can be found in Section 2 of the supplementary document. Moreover, in Section 4 we describe how A_m can be efficiently computed for all nodes m with a single pass over the tree.

As a final remark, we considered also an alternative optimization procedure to SGD, namely Resilient Back-Propagation (RPROP) [27], which automatically adapts a specific learning rate for each parameter based on the sign change of its risk partial derivative over the last iteration.

3.2. Learning Prediction Nodes

Given the update rules for the decision function parameters Θ from the previous subsection, we now consider the problem of minimizing (5) with respect to $\boldsymbol{\pi}$ when Θ is fixed, *i.e.*

$$\min_{\boldsymbol{\pi}} R(\Theta, \boldsymbol{\pi}; \mathcal{T}) \,. \tag{10}$$

This is a *convex* optimization problem and a global solution can be easily recovered. A similar problem has been encountered in the context of decision trees in [28], but only at the level of a single node. In our case, however, the whole tree is taken into account, and we are jointly estimating *all* the leaf predictions.

In order to compute a global minimizer of (10) we propose the following iterative scheme:

$$\pi_{\ell y}^{(t+1)} = \frac{1}{Z_{\ell}^{(t)}} \sum_{(\boldsymbol{x}, y') \in \mathcal{T}} \frac{\mathbb{1}_{y=y'} \, \pi_{\ell y}^{(t)} \mu_{\ell}(\boldsymbol{x}|\Theta)}{\mathbb{P}_T[y|\boldsymbol{x}, \Theta, \boldsymbol{\pi}^{(t)}]} \,, \tag{11}$$

for all $\ell \in \mathcal{L}$ and $y \in \mathcal{Y}$, where $Z_{\ell}^{(t)}$ is a normalizing factor ensuring that $\sum_y \pi_{\ell y}^{(t+1)} = 1$. The starting point $\boldsymbol{\pi}^{(0)}$ can be arbitrary as long as every element is positive. A typical choice is to start from the uniform distribution in all leaves, *i.e.* $\pi_{\ell y}^{(0)} = |\mathcal{Y}|^{-1}$. It is interesting to note that the update rule in (11) is step-size free and it guarantees a strict decrease of the risk at each update until a fixed-point is reached (see proof in supplementary material).

As opposed to the update strategy for Θ, which is based on mini-batches, we adopt an offline learning approach to obtain a more reliable estimate of $\boldsymbol{\pi}$, because suboptimal predictions in the leaves have a strong impact on the final prediction. Moreover, we interleave the update of $\boldsymbol{\pi}$ with a whole epoch of stochastic updates of Θ as described in the previous subsection.

3.3. Learning a Forest

So far we have dealt with a single decision tree setting. Now, we consider an ensemble of trees $\mathcal{F}$, where all trees can possibly share same parameters in Θ, but each tree can have a different structure with a different set of decision functions (still defined as in (3)), and independent leaf predictions $\boldsymbol{\pi}$.

Since each tree in forest $\mathcal{F}$ has its own set of leaf parameters $\boldsymbol{\pi}$, we can update the prediction nodes of each tree independently as described in Subsection 3.2, given the current estimate of Θ.

As for Θ, instead, we randomly select a tree in $\mathcal{F}$ for each mini-batch and then we proceed as detailed in Subsection 3.1 for the SGD update. This strategy somewhat resembles the basic idea of Dropout [34], where each SGD update is potentially applied to a different network topology, which is sampled according to a specific distribution. In addition, updating individual trees instead of the entire forest reduces the computational load during training.

During test time, as shown in (4), the prediction delivered by each tree is averaged to produce the final outcome.

3.4. Summary of the Learning Procedure

The learning procedure is summarized in Algorithm 1. We start with a random initialization of the decision nodes parameters Θ and iterate the learning procedure for a predetermined number of epochs, given a training set $\mathcal{T}$. At each epoch, we initially obtain an estimation of the prediction node parameters $\boldsymbol{\pi}$ given the actual value of Θ by running the iterative scheme in (11), starting from the uniform distribution in each leaf, *i.e.* $\pi_{\ell y}^{(0)} = |\mathcal{Y}|^{-1}$. Then we split the training set into a random sequence of mini-batches and we perform for each mini-batch a SGD update of Θ as in (7). After each epoch we might eventually change the learning rate according to pre-determined schedules.

More details about the computation of some tree-specific terms are given in the next section.

Algorithm 1 Learning trees by back-propagation

Require: $\mathcal{T}$: training set, nEpochs
1: random initialization of Θ
2: **for all** $i \in \{1, \ldots,$ nEpochs$\}$ **do**
3: Compute $\boldsymbol{\pi}$ by iterating (11)
4: break $\mathcal{T}$ into a set of random mini-batches
5: **for all** $\mathcal{B}$: mini-batch from $\mathcal{T}$ **do**
6: Update $\boldsymbol{\Theta}$ by SGD step in (7)
7: **end for**
8: **end for**

4. Implementation Notes

4.1. Decision Nodes

We have defined decision functions d_n in terms of real-valued functions $f_n(\cdot; \Theta)$, which are not necessarily independent, but coupled through the shared parametrization Θ. Our intention is to endow the trees with feature learning capabilities by embedding functions f_n within a deep convolutional neural network with parameters Θ. In the specific, we can regard each function f_n as a linear output unit of a deep network that will be turned into a probabilistic routing decision by the action of d_n, which applies a sigmoid activation to obtain a response in the $[0, 1]$ range. Fig. 2 provides a schematic illustration of this idea, showing how

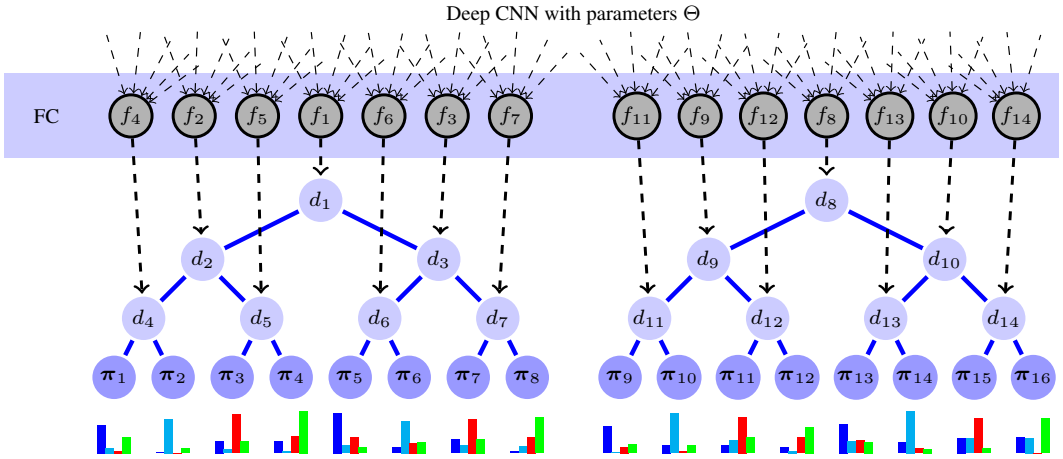

Figure 2. Illustration how to implement a deep neural decision forest (dNDF). Top: Deep CNN with variable number of layers, subsumed via parameters Θ. FC block: Fully Connected layer used to provide functions $f_n(\cdot;\Theta)$ (here: inner products), described in Equ. (3). Each output of f_n is brought in correspondence with a split node in a tree, eventually producing the routing (split) decisions $d_n(\boldsymbol{x}) = \sigma(f_n(\boldsymbol{x}))$. The order of the assignments of output units to decision nodes can be arbitrary (the one we show allows a simple visualization). The circles at bottom correspond to leaf nodes, holding probability distributions $\boldsymbol{\pi}_\ell$ as a result from solving the convex optimization problem defined in Equ. (10).

decision nodes can be implemented by using typically available fully-connected (or inner-product) and sigmoid layers in DNN frameworks like Caffe or MatConvNet. Easy to see, the number of split nodes is determined by the number of output nodes of the preceding fully-connected layer.

Under the proposed construction, the output units of the deep network are therefore not directly delivering the final predictions, *e.g.* through a Softmax layer, but each unit is responsible for driving the decision of a node in the forest. Indeed, during the forward pass through the deep network, a data sample $\boldsymbol{x}$ produces soft activations of the routing decisions of the tree that induce via the routing function a mixture of leaf predictions as per (1), which will form the final output. Finally, please note that by assuming linear and independent (via separate parametrizations) functions $f_n(\boldsymbol{x};\boldsymbol{\theta}_n) = \boldsymbol{\theta}_n^\top \boldsymbol{x}$, we recover a model similar to oblique forests [13].

4.2. Routing Function

The computation of the routing function μ_ℓ can be carried out by traversing the tree once. Let $\top \in \mathcal{N}$ be the root node and for each node $n \in \mathcal{N}$ let n_l and n_r denote its left and right child, respectively. We start from the root by setting $\mu_\top = 1$ and for each node $n \in \mathcal{N}$ that we visit in breadth-first order we set $\mu_{n_l} = d_n(\boldsymbol{x};\Theta)\mu_n$ and $\mu_{n_r} = \bar{d}_n(\boldsymbol{x};\Theta)\mu_n$. At the end, we can read from the leaves the desired values of the routing function.

4.3. Learning Decision Nodes

The forward pass of the back-propagation algorithm precomputes the values of the routing function $\mu_\ell(\boldsymbol{x};\Theta)$ and the value of the tree prediction $\mathbb{P}_T[y|\boldsymbol{x},\Theta,\boldsymbol{\pi}]$ for each sample $(\boldsymbol{x}, y)$ in the mini-batch $\mathcal{B}$. The backward pass requires the computation of the gradient term in (9) for each sample $(\boldsymbol{x}, y)$ in the mini-batch. This can be carried out by a single, bottom-up tree traversal. We start by setting

$$A_\ell = \frac{\pi_{\ell y}\mu_\ell(\boldsymbol{x};\Theta)}{\mathbb{P}_T[y|\boldsymbol{x},\Theta,\boldsymbol{\pi}]}$$

for each $\ell \in \mathcal{L}$. Then we visit the tree in reversed breadth-first order (bottom-up). Once in a node $n \in \mathcal{N}$, we can compute the partial derivative in (9) since we can read A_{n_l} and A_{n_r} from the children, and we set $A_n = A_{n_l} + A_{n_r}$, which will be required by the parent node.

4.4. Learning Prediction Nodes

Before starting the iterations in (11), we precomputed $\mu_\ell(\boldsymbol{x};\Theta)$ for each $\ell \in \mathcal{L}$ and for each sample $\boldsymbol{x}$ in the training set, as detailed in Subsection 4.2. The iterative scheme requires few iterations to converge to a solution with an acceptable accuracy (20 iterations were enough for all our experiments).

5. Experiments

Our experiments illustrate both, the performance of shallow neural decision forests (sNDFs) as standalone classifiers, as well as their effect when used as classifiers in deep, convolutional neural networks (dNDF). To this end, we evaluate our proposed classifiers on diverse datasets, covering a broad range of classification tasks (ranging from simple binary classification of synthetically generated data up to large-scale image recognition on the 1000-class ImageNet dataset).

	G50c [33]	Letter [10]	USPS [14]	MNIST [20]	Char74k[8]
# Train Samples	50	16000	7291	60000	66707
# Test Samples	500	4000	2007	10000	7400
# Classes	2	26	10	10	62
# Input dimensions	50	16	256	784	64
Alternating Decision Forest (ADF) [30]	18.71±1.27	3.52±0.17	5.59±0.16	2.71±0.10	16.67±0.21
Shallow Neural Decision Forest (sNDF)	17.4±1.52	**2.92±0.17**	**5.01±0.24**	2.8±0.12	**16.04±0.20**
Tree input features	10 (random)	8 (random)	10x10 patches	15x15 patches	10 (random)
Depth	5	10	10	10	12
Number of trees	50	70	100	80	200
Batch size	25	500	250	1000	1000

Table 1. Comparison of alternating decision forests (ADF) to shallow neural decision forests (sNDFs, no hidden layers) on selected standard machine learning datasets. Top: Details about datasets. Middle: Average error [%] (with corresponding standard deviation) obtained from 10 repetitions of the experiment. Bottom: Details about the parametrization of our model.

5.1. Comparison of sNDFs to Forest Classifiers

We first compared sNDFs against state-of-the-art in terms of stand-alone, off-the-shelf forest ensembles. In order to have a fair comparison, our classifier is built *without hidden layers*, *i.e.* we consider feature mappings having the simple form $f_n(\boldsymbol{x};\boldsymbol{\theta}_n) = \boldsymbol{\theta}_n^\top \boldsymbol{x}$. We used the 5 datasets in [30] to compare the performance of sNDFs to that of Alternating Decision Forests (ADF). The details of this experiment are summarized in Tab. 1. For ADF, we provide results reported in their paper and we use their reported maximal tree depth and forest size as an upper bound on the size of our models. Essentially, for each of the datasets, all our trees are less deep and there are fewer of them than in the corresponding ADF models. We used ensembles of different sizes depending on the size of the dataset and the complexity of the learning tasks. In all cases, we use RPROP and the recommended hyper-parameters of the original publication [27] for split node parameter optimization. We report the average error with standard deviations resulting from 10 repetitions of the experiment. Overall, we outperform ADF, though significant results, with p-values less than 0.05, were obtained for the `Letter`, `USPS` and `Char74k` datasets.

5.2. Improving Performance with dNDF

In the following experiments we integrated our novel forest classifiers in end-to-end image classification pipelines, using multiple convolutional layers for representation learning as typically done in deep learning systems.

5.2.1 MNIST

We used the MatConvNet library [37] and their reference implementation of LeNet-5 [21], for building an end-to-end digit classification system on MNIST training data, replacing the conventionally used Softmax layer by our forest. The baseline yields an error of 0.9% on test data, which we obtained by re-running the provided example architecture with given settings for optimization, hyper-parameters, *etc*. By using our proposed dNDF on top of LeNet-5, each decision function being driven by an output unit fully connected to the last hidden layer of the CNN, we can reduce the classification error to 0.7%. The ensemble size was fixed to 10 trees, each with a depth of 5. Please note the positive effect on MNIST performance compared to Section 5.1 when spending additional layers on representation learning.

5.2.2 ImageNet

ImageNet [29] is a benchmark for large-scale image recognition tasks and its images are assigned to a single out of 1000 possible ground truth labels. The dataset contains ≈1.2M training images, 50.000 validation images and 100.000 test images with average dimensionality of 482x415 pixels. Training and validation data are publicly available and we followed the commonly agreed protocol by reporting *Top5-Errors* on validation data. The GoogLeNet architecture [36], which has a reported Top5-Error of 10.07% when used in a single-model, single-crop setting (see first row in Tab. 3 in [36]) served as basis for our experiments. It uses 3 Softmax layers at different stages of the network to encourage the construction of informative features, due to its very deep architecture. Each of these Softmax layers gets their input from a Fully Connected (FC) layer, built on top of an Average Pool layer, which in turn is built on top of a corresponding Concat layer. Let DC0, DC1 and DC2 be the Concat layers preceding each of the Softmax layers in GoogLeNet. Let AvgPool0, AvgPool1 and AvgPool2 be the Average Pool layers preceding these Softmax layers. To avoid problems with propagation of gradients given the depth of the network and in order to provide the final classification layers with the features obtained in the early stages of the pipeline, we have also supplied DC0

	GoogLeNet [36]						GoogLeNet⋆	dNDF.NET		
# Models	1			7			1	1		7
# Crops	1	10	144	1	10	144	1	1	10	1
Top5-Errors	10.07%	9.15%	7.89%	8.09%	7.62%	6.67%	10.02%	7.84%	7.08%	**6.38%**

Table 2. Top5-Errors obtained on ImageNet validation data, comparing our dNDF.NET to GoogLeNet(⋆).

as input to AvgPool1 and AvgPool2 and DC1 as input to AvgPool2. We have implemented this modified network using the Distributed (Deep) Machine Learning Common (DMLC) library [22][2] and dub it *GoogLeNet⋆*. Its single-model, single crop Top5-Error is 10.02% (when trained with SGD, 0.9 momentum, fixed learning rate schedule, decreasing the learning rate by 4% every 8 epochs and mini-batches composed of 50 images).

In order to obtain a *Deep Neural Decision Forest* architecture coined dNDF.NET, we have replaced each Softmax layer from GoogLeNet⋆ with a forest consisting of 10 trees (each fixed to depth 15), resulting in a total number of 30 trees. We refer to the individual forests as dNDF_0 (closest to raw input), dNDF_1 (replacing middle loss layer in GoogLeNet⋆) and dNDF_2 (as terminal layer). We provide a visualization for our dNDF.NET architecture in the supplementary document. Following the implementation guideline in Subsection 4.1, we randomly selected 500 output dimensions of the respectively preceding layers in GoogLeNet⋆ for each decision function f_n. In such a way, a single FC layer with #trees $\times$ #split nodes/tree output units provides all the split node inputs per dNDF_x. The resulting architecture was implemented in DMLC as well, and we trained the network for 1000 epochs using (mini-) batches composed of 100.000 images (which was feasible due to distribution of the computational load to a cluster of 52 CPUs and 12 hosts, where each host is equipped with a NVIDIA Tesla K40 GPU).

For posterior learning, we only update the leaf node predictions of the tree that also receives split node parameter updates, *i.e*. the randomly selected one as described in Subsection 3.3. To improve computational efficiency, we consider only the samples of the current mini-batch for posterior learning, while *all* the training data could be used in principle. However, since we use mini-batches composed of 100.000 samples, we can approximate the training set sufficiently well while simultaneously introducing a positive, regularizing effect.

Tab. 2 provides a summary of Top5-Errors on validation data for our proposed dNDF.NET against GoogLeNet and GoogLeNet⋆. We ascribe the improvements on the single crop, single model setting (Top5-Error of only 7.84%) to our proposed approach, as the only architectural difference to GoogLeNet⋆ (Top5-Error of 10.02%) is deploying our dNDFs. By using an ensemble of 7 dNDF.NETs (still single crop inputs), we can improve further and obtain a Top5-Error of 6.38%, which is better than the best result of 6.67%, obtained with 7 GoogLeNets using 144 crops per image [36]. Next, we discuss some operational characteristics of our single-model, single-crop setting.

Evaluation of tree nodes Intuitively, a tree-structured classifier aims to produce *pure* leaf node distributions. This means that the training error is reduced by (repeatedly) partitioning the input space in a way such that it correlates with target classes in $\mathcal{Y}$.

Analyzing the outputs of decision functions f_n is informative about the routing uncertainties for a given sample $\boldsymbol{x}$, as it traverses the tree(s). In Fig. 3, we show histograms of all available split node outputs of our three forests (*i.e*. dNDF_0, dNDF_1, dNDF_2) for all samples of the validation set after running for 100, 500 and 1000 epochs over the training data. The leftmost histogram (after 100 training epochs) shows the highest uncertainty about the routing direction, *i.e*. the split decisions are not yet very crisp such that a sample will be routed to many leaf nodes. As training progresses (middle and right plots after 500 and 1000 epochs), we can see how the distributions get very peaked at 0 and 1 (*i.e*. samples are routed either to the left or right child with low uncertainty), respectively. As a result, the samples will only be routed to a small subset of available leaf nodes with reasonably high probability. In other words, most available leaves will never be reached from a sample-centric view and therefore only a small number of overall paths needs to be evaluated at test time. As part of future work and in order to decrease computational load, we plan to route samples along the trees by sampling from these split distributions, rather than sending them to every leaf node.

To assess the quality of the resulting leaf posterior distributions obtained from the global optimization procedure, we illustrate how the mean leaf entropy develops as a function of training epochs (see Fig. 4). To this end, we randomly selected 1024 leaves from all available ones per tree and computed their mean entropy after each epoch. The highest entropy would result from a uniform distribution and is $\approx$9.96 bits for a 1000-class problem. Instead, we want to obtain highly peaked distributions for the leaf predictors, leading to low entropy. Indeed, the average entropy decreases as training progresses, confirming the efficacy of our proposed leaf node parameter learning approach.

[2] https://github.com/dmlc/cxxnet.git

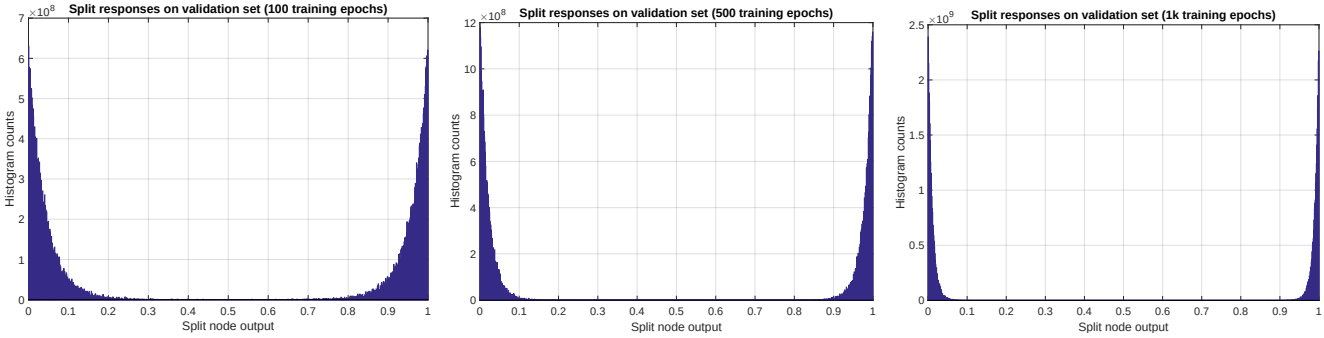

Figure 3. Histograms over all split node responses of all three forests in dNDF**.**NET on ImageNet validation data after accomplishing 100 (left), 500 (middle) and 1000 (right) epochs over training data. As training progresses, the split node outputs approach 0 or 1 which corresponds to eliminating routing uncertainty of samples when being propagated through the trees.

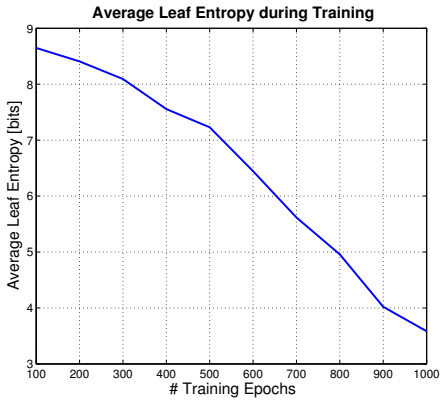

Figure 4. Average leaf entropy development as a function of training epochs in dNDF**.**NET.

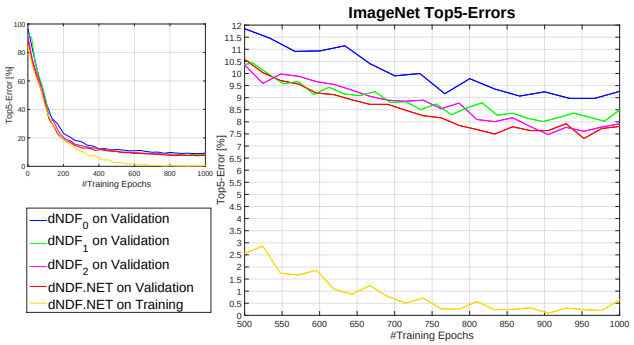

Figure 5. Top5-Error plots for individual dNDF$_x$ used in dNDF**.**NET as well as their joint ensemble errors. Left: Plot over all 1000 training epochs. Right: Zoomed version of left plot, showing Top5-Errors from 0–12% between training epochs 500-1000.

Evaluation of model performance In Fig. 5 we show the development of Top5-Errors for each dNDF$_x$ in dNDF**.**NET as well as their ensemble performances as a function of training epochs. The left plot shows the development over all training epochs (1000 in total) while the right plot is a zoomed view from epochs 500 to 1000 and Top5-Errors 0–12%. As expected, dNDF$_0$ (which is closest to the input layer) performs worse than dNDF$_2$, which constitutes the final layer of dNDF**.**NET, however, only by 1.34%. Consequently, the computational load between dNDF$_0$ and dNDF$_2$ could be traded for a degradation of only 1.34% in accuracy during inference. Conversely, taking the mean over all three dNDFs yields the lowest Top5-Error of 7.84% after 1000 epochs over training data.

6. Conclusions

In this paper we have shown how to model and train stochastic, differentiable decision trees, usable as alternative classifiers for end-to-end learning in (deep) convolutional networks. Prevailing approaches for decision tree training typically operate in a greedy and local manner, making representation learning impossible. To overcome this problem, we introduced stochastic routing for decision trees, enabling split node parameter learning via back-propagation. Moreover, we showed how to populate leaf nodes with their optimal predictors, given the current state of the tree/underlying network. We have successfully validated our new decision forest model as stand-alone classifier on standard machine learning datasets and surpass state-of-the-art performance on ImageNet when integrating them in the GoogLeNet architecture, without any form of dataset augmentation.

Acknowledgments Peter and Samuel were partially supported by Novartis Pharmaceuticals and the ASSESS MS project. Madalina was partially supported by the National Science Foundation under Award 1320347 and by the Defense Advanced Research Projects Agency under Contract FA8750-12-2-0324. We thank Alex Smola and Mu Li for granting us access to run ImageNet experiments on their server infrastructure and Darko Zikic for fruitful discussions.

References

[1] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *(NC)*, 9(7):1545–1588, 1997. 1

[2] C. M. Bishop and M. Svensén. Bayesian hierarchical mixtures of experts. In *Proc. of Conference on Uncertainty in Artificial Intelligence*, pages 57–64, 2003. 2

[3] A. Bosch, A. Zisserman, and X. Muñoz. Image classification using random forests and ferns. In *(ICCV)*, 2007. 1

[4] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. 1

[5] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla. Segmentation and recognition using structure from motion point clouds. In *(ECCV)*. Springer, 2008. 1

[6] R. Caruana, N. Karampatziakis, and A. Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *(ICML)*, pages 96–103, 2008. 1

[7] A. Criminisi and J. Shotton. *Decision Forests in Computer Vision and Medical Image Analysis*. Springer, 2013. 1, 2

[8] T. E. de Campos, B. R. Babu, and M. Varma. Character recognition in natural images. In *Proceedings of the International Conference on Computer Vision Theory and Applications, Lisbon, Portugal*, February 2009. 6

[9] A. K. L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *(CVPR)*, 2015. 1

[10] P. W. Frey and D. J. Slate. Letter recognition using holland-style adaptive classifiers. *(ML)*, 6(2), Mar. 1991. 6

[11] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, 2009. 1

[12] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. 1

[13] D. Heath, S. Kasif, and S. Salzberg. Induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2(2):1–32, 1993. 3, 5

[14] J. J. Hull. A database for handwritten text recognition research. *(PAMI)*, 16(5):550–554, 1994. 6

[15] J. Jancsary, S. Nowozin, and C. Rother. Loss-specific training of non-parametric image restoration models: A new state of the art. In *(ECCV)*, 2012. 2

[16] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 2

[17] M. I. Jordan. Hierarchical mixtures of experts and the em algorithm. *(NC)*, 6:181–214, 1994. 2

[18] P. Kontschieder, P. Kohli, J. Shotton, and A. Criminisi. GeoF: Geodesic forests for learning coupled predictors. In *(CVPR)*, pages 65–72, 2013. 1

[19] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *(NIPS)*, 2012. 1

[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998. 6

[21] Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and M. K., editors, *Neural Networks: Tricks of the trade*. Springer, 1998. 6

[22] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, 2014. 7

[23] M. Lin, Q. Chen, and S. Yan. Network in network. *CoRR*, abs/1312.4400, 2013. 1

[24] A. Montillo, J. Tu, J. Shotton, J. Winn, J. E. Iglesias, D. N. Metaxas, and A. Criminisi. Entangled forests and differentiable information gain maximization. In *Decision Forests in Computer Vision and Medical Image Analysis*. Springer, 2013. 1, 2

[25] S. Ren, X. Cao, Y. Wei, and J. Sun. Global refinement of random forest. In *(CVPR)*, 2015. 2

[26] D. L. Richmond, D. Kainmueller, M. Y. Yang, E. W. Myers, and C. Rother. Relating cascaded random forests to deep convolutional neural networks for semantic segmentation. *CoRR*, abs/1507.07583, 2015. 2

[27] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *IEEE Conf. on Neural Networks*, 1993. 4, 6

[28] S. Rota Bulò and P. Kontschieder. Neural decision forests for semantic image labelling. In *(CVPR)*, 2014. 1, 4

[29] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 2014. 6

[30] S. Schulter, P. Wohlhart, C. Leistner, A. Saffari, P. M. Roth, and H. Bischof. Alternating decision forests. In *(CVPR)*, 2013. 2, 6

[31] I. Sethi. Entropy nets: from decision trees to neural networks. *Proceedings of the IEEE*, 78(10), 1990. 2

[32] J. Shotton, R. Girshick, A. Fitzgibbon, T. Sharp, M. Cook, M. Finocchio, R. Moore, P. Kohli, A. Criminisi, A. Kipman, and A. Blake. Efficient human pose estimation from single depth images. *(PAMI)*, 2013. 1

[33] V. Sindhwani, P. Niyogi, and M. Belkin. Beyond the point cloud: From transductive to semi-supervised learning. In *(ICML)*, pages 824–831. ACM, 2005. 6

[34] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. 4

[35] A. Suárez and J. F. Lutsko. Globally optimal fuzzy decision trees for classification and regression. *(PAMI)*, 21(12):1297–1311, 1999. 2

[36] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. 1, 6, 7

[37] A. Vedaldi and K. Lenc. Matconvnet – convolutional neural networks for matlab. *CoRR*, abs/1412.4564, 2014. 2, 6

[38] D. Yu and L. Deng. *Automatic Speech Recognition: A Deep Learning Approach*. Springer, 2014. 1

Deep Neural Decision Forests

Peter Kontschieder[1] Madalina Fiterau[2] Antonio Criminisi[1] Samuel Rota Bulò[1,3]

Microsoft Research[1] Cambridge, UK
Carnegie Mellon University[2] Pittsburgh, PA
Fondazione Bruno Kessler[3] Trento, Italy

Supplementary Material

This document provides the following supplementary contributions:

- Additional details regarding the ImageNet experiment (see Section 1).
- Detailed derivations for the gradient term depending on the decision tree splits (see Section 2).
- The proof that our update rule for the leaf predictions $\boldsymbol{\pi}$ in Equ. (11) of our main ICCV paper monotonically decreases the risk R until a fixed point is reached (see Section 3).

1. ImageNet experiment: GoogLeNet vs. dNDF.NET architectures

This section provides additional description for the ImageNet experiment [3]. In particular, we describe and illustrate the changes we made to GoogLeNet [4] to obtain our proposed dNDF**.**NET architecture, using deep neural decision forests (dNDFs) as classifiers. The GoogLeNet architecture we have used as basis for our experiments (see left illustration in Fig. 1, taken from [4]) has a reported Top5-Error of 10.07%, when used in a single-model, single-crop setting (see first row in Tab. 3 of [4]).

Fig. 1 (right illustration) shows that we have introduced two different modifications with respect to the original GoogLeNet architecture. First, we have connected the outputs of the Concat layers to the inputs of the AveragePool layers (as described in the main paper), visualized by red arrows in the plot. The resulting, modified baseline network is dubbed GoogLeNet$\star$ and achieves a Top5-Error of 10.02% when using conventional SoftMax layers as in the original network. The implementation yielding this score was realized in the Distributed (Deep) Machine Learning Common (DMLC) library [2, 1][1], using resized images with dimensionality 100x100 as described in [2]. The training used the standard settings for GoogLeNet, stochastic gradient descent with 0.9 momentum, fixed learning rate schedule, decreasing the learning rate by 4% every 8 epochs. We trained GoogLeNet$\star$ with mini-batches composed of 50 images.

In order to obtain a *Deep Neural Decision Forest* architecture coined dNDF**.**NET, we have replaced each Softmax layer from GoogLeNet$\star$ with a forest consisting of 10 trees (each fixed to depth 15), resulting in a total number of 30 trees. For our architecture, which we implemented in DMLC as well, we trained the network for 1000 epochs using mini-batches composed of 100.000 images. This is feasible due to distribution of the computational load to a cluster of 52 CPUs and 12 hosts, where each host is equipped with a NVIDIA Tesla K40 GPU.

We refer to the individual forests as dNDF_0, dNDF_1 and dNDF_2, where dNDF_0 is closest to the input layer and dNDF_2 is the final (last) layer in the architecture. Each tree is a balanced and fixed depth 15 tree, which means that the total number of per-tree split nodes is $2^{15} - 1 = 32.767$ and the number of leaf nodes is $2^{15} = 32.768$.

[1] https://github.com/dmlc/cxxnet.git

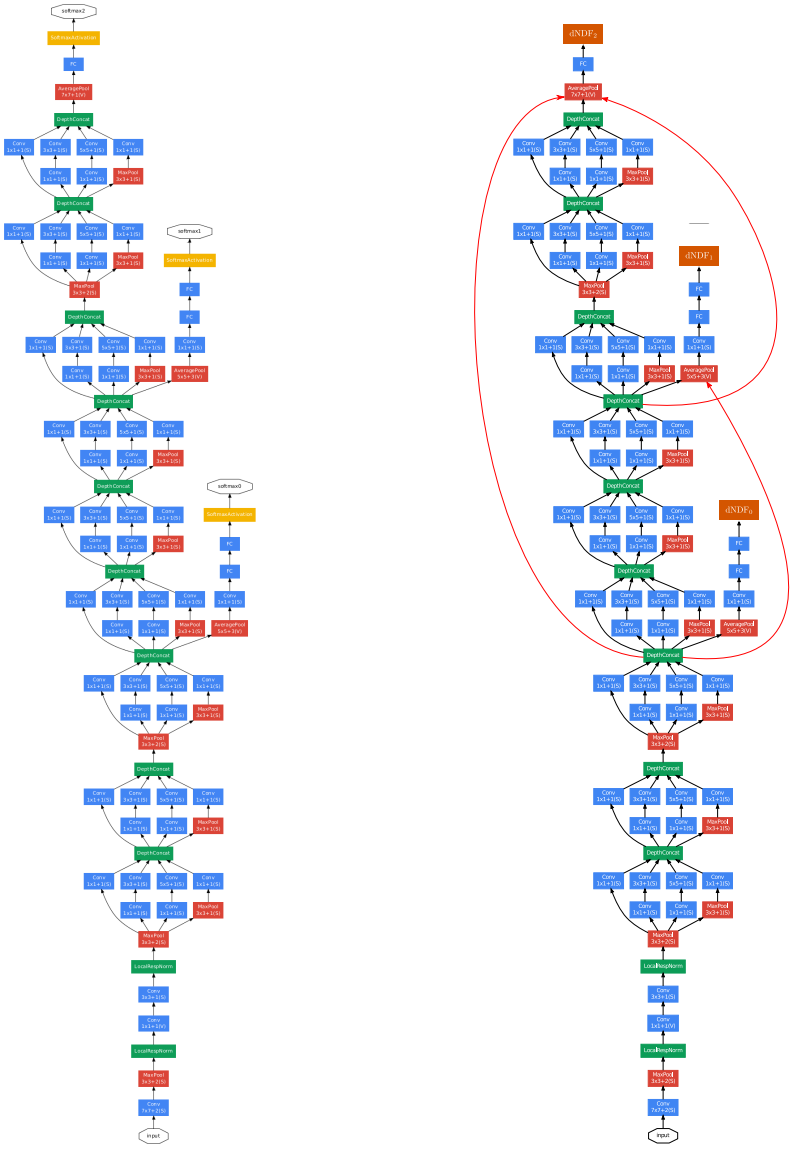

Figure 1. Left: Original GoogLeNet architecture proposed in [4]. Right: The modifications we brought to the GoogLeNet architecture resulting in dNDF.NET – our proposed model using dNDFs as final classifiers. Best viewed with digital zoom.

2. Split function gradient term derivations

Given the definitions for our split decision functions $d_n(\boldsymbol{x};\Theta)$ and the log-loss $L(\Theta,\boldsymbol{\pi};\boldsymbol{x},y)$ (see Equ. (3) and Equ. (6) in the main paper, respectively), we can derive the gradient term in Equ. (9) of the main paper as follows:

$$\begin{aligned}\frac{\partial L(\Theta,\boldsymbol{\pi};\boldsymbol{x},y)}{\partial f_n(\boldsymbol{x};\Theta)} &= \sum_{\ell\in\mathcal{L}} \frac{\partial L(\Theta,\boldsymbol{\pi};\boldsymbol{x},y)}{\partial \mu_\ell(\boldsymbol{x};\Theta)} \frac{\partial \mu_\ell(\boldsymbol{x};\Theta)}{\partial f_n(\boldsymbol{x};\Theta)} \\ &= -\sum_{\ell\in\mathcal{L}} \frac{\pi_{\ell y}}{\mathbb{P}_T[y|\boldsymbol{x},\Theta,\boldsymbol{\pi}]} \frac{\partial \mu_\ell(\boldsymbol{x};\Theta)}{\partial f_n(\boldsymbol{x};\Theta)} \\ &= -\sum_{\ell\in\mathcal{L}} \frac{\pi_{\ell y}\mu_\ell(\boldsymbol{x};\Theta)}{\mathbb{P}_T[y|\boldsymbol{x},\Theta,\boldsymbol{\pi}]} \frac{\partial \log \mu_\ell(\boldsymbol{x};\Theta)}{\partial f_n(\boldsymbol{x};\Theta)}\,,\end{aligned}$$

where

$$\begin{aligned}\frac{\partial \log \mu_\ell(\boldsymbol{x};\Theta)}{\partial f_n(\boldsymbol{x};\Theta)} &= \mathbb{1}_{\ell \swarrow n} \frac{\partial \log d_n(\boldsymbol{x};\Theta)}{\partial f_n(\boldsymbol{x};\Theta)} \\ &\quad + \mathbb{1}_{n \searrow \ell} \frac{\partial \log \bar{d}_n(\boldsymbol{x};\Theta)}{\partial f_n(\boldsymbol{x};\Theta)} \\ &= \mathbb{1}_{\ell \swarrow n} \bar{d}_n(\boldsymbol{x};\Theta) - \mathbb{1}_{n \searrow \ell} d_n(\boldsymbol{x};\Theta)\,.\end{aligned}$$

By substituting the latter in the previous formula we get

$$\begin{aligned}\frac{\partial L(\Theta,\boldsymbol{\pi};\boldsymbol{x},y)}{\partial f_n(\boldsymbol{x};\Theta)} &= -\sum_{\ell\in\mathcal{L}} \mathbb{1}_{\ell \swarrow n} \frac{\pi_{\ell y}\mu_\ell(\boldsymbol{x};\Theta)}{\mathbb{P}_T[y|\boldsymbol{x},\Theta,\boldsymbol{\pi}]} \bar{d}_n(\boldsymbol{x};\Theta) \\ &\quad + \sum_{\ell\in\mathcal{L}} \mathbb{1}_{n \searrow \ell} \frac{\pi_{\ell y}\mu_\ell(\boldsymbol{x};\Theta)}{\mathbb{P}_T[y|\boldsymbol{x},\Theta,\boldsymbol{\pi}]} d_n(\boldsymbol{x};\Theta) \\ &= -\sum_{\ell\in\mathcal{L}_{n_l}} \frac{\pi_{\ell y}\mu_\ell(\boldsymbol{x};\Theta)}{\mathbb{P}_T[y|\boldsymbol{x},\Theta,\boldsymbol{\pi}]} \bar{d}_n(\boldsymbol{x};\Theta) \\ &\quad + \sum_{\ell\in\mathcal{L}_{n_r}} \frac{\pi_{\ell y}\mu_\ell(\boldsymbol{x};\Theta)}{\mathbb{P}_T[y|\boldsymbol{x},\Theta,\boldsymbol{\pi}]} d_n(\boldsymbol{x};\Theta) \\ &= d_n(\boldsymbol{x};\Theta) A_{n_r} - \bar{d}_n(\boldsymbol{x};\Theta) A_{n_l}\,.\end{aligned}$$

3. Proof of update rule for $\boldsymbol{\pi}$

Theorem 1. *Consider a tree with parameters Θ and $\boldsymbol{\pi}$ and let*

$$\hat{\pi}_{\ell y} = \frac{1}{Z_\ell} \sum_{(\boldsymbol{x},y')\in\mathcal{T}} \mathbb{1}_{y=y'} \frac{\pi_{\ell y}\mu_\ell(\boldsymbol{x};\Theta)}{\mathbb{P}_T[y|\boldsymbol{x};\Theta,\boldsymbol{\pi}]}\,, \qquad \textit{for all } (\ell,y)\in\mathcal{L}\times\mathcal{Y}\,, \tag{12}$$

where Z_ℓ is the normalizing factor ensuring that $\hat{\boldsymbol{\pi}}_\ell = (\hat{\pi}_{\ell y})_{y\in\mathcal{Y}}$ is a probability distribution. In other terms, we assume $\hat{\pi}_{\ell y}$ to be the result of an update step as per (11) of our ICCV contribution. The following holds:

$$R(\Theta,\boldsymbol{\pi};\mathcal{T}) \geq R(\Theta,\hat{\boldsymbol{\pi}};\mathcal{T})$$

with equality if and only if $\hat{\boldsymbol{\pi}} = \boldsymbol{\pi}$, where R is the risk defined in (5) of our ICCV contribution, and $\boldsymbol{\pi} = (\boldsymbol{\pi}_\ell)_{\ell\in\mathcal{L}}$.

Proof. Consider the following auxiliary function:

$$\phi(\boldsymbol{\pi},\overline{\boldsymbol{\pi}}) = R(\Theta,\overline{\boldsymbol{\pi}};\mathcal{T}) - \frac{1}{|\mathcal{T}|} \sum_{(\boldsymbol{x},y)\in\mathcal{T}} \sum_{\ell\in\mathcal{L}} \xi_\ell(\overline{\boldsymbol{\pi}};\boldsymbol{x},y) \log\left(\frac{\pi_{\ell y}}{\overline{\pi}_{\ell y}}\right)\,,$$

where

$$\xi_\ell(\boldsymbol{\pi};\boldsymbol{x},y) = \frac{\pi_{\ell y}\mu_\ell(\boldsymbol{x};\Theta)}{\mathbb{P}_T[y|\boldsymbol{x};\Theta,\boldsymbol{\pi}]}\,.$$

and $\mathbb{P}_T[y|\boldsymbol{x},\Theta,\boldsymbol{\pi}]$ is defined as per (1) of our ICCV contribution. Note that $\phi(\boldsymbol{\pi},\boldsymbol{\pi}) = R(\Theta,\boldsymbol{\pi};\mathcal{T})$ holds for any $\boldsymbol{\pi}$, for the logarithm term in ϕ nullifies. Moreover, $\phi(\boldsymbol{\pi},\overline{\boldsymbol{\pi}}) \geq R(\Theta,\boldsymbol{\pi};\mathcal{T})$ holds for any $\boldsymbol{\pi}$ and $\overline{\boldsymbol{\pi}}$. This can be seen by applying

Jensen's inequality and with few algebraic manipulations:

$$
\begin{aligned}
R(\Theta, \boldsymbol{\pi}; \mathcal{T}) &= -\frac{1}{|\mathcal{T}|} \sum_{(\boldsymbol{x},y)\in\mathcal{T}} \log\left(\sum_{\ell\in\mathcal{L}} \pi_{\ell y}\mu_\ell(\boldsymbol{x};\Theta)\right) \\
&\leq -\frac{1}{|\mathcal{T}|} \sum_{(\boldsymbol{x},y)\in\mathcal{T}} \sum_{\ell\in\mathcal{L}} \xi_\ell(\overline{\boldsymbol{\pi}};\boldsymbol{x},y) \log\left(\frac{\pi_{\ell y}\mu_\ell(\boldsymbol{x};\Theta)}{\xi_\ell(\overline{\boldsymbol{\pi}};\boldsymbol{x},y)}\right) \qquad \text{(by Jensen's inequality)} \\
&= -\frac{1}{|\mathcal{T}|} \sum_{(\boldsymbol{x},y)\in\mathcal{T}} \sum_{\ell\in\mathcal{L}} \xi_\ell(\overline{\boldsymbol{\pi}};\boldsymbol{x},y) \left[\log\left(\frac{\pi_{\ell y}}{\overline{\pi}_{\ell y}}\right) + \log \mathbb{P}_T[y|\boldsymbol{x},\Theta,\overline{\boldsymbol{\pi}}]\right] \\
&= R(\Theta,\overline{\boldsymbol{\pi}};\mathcal{T}) - \frac{1}{|\mathcal{T}|} \sum_{(\boldsymbol{x},y)\in\mathcal{T}} \sum_{\ell\in\mathcal{L}} \xi_\ell(\overline{\boldsymbol{\pi}};\boldsymbol{x},y) \log\left(\frac{\pi_{\ell y}}{\overline{\pi}_{\ell y}}\right) = \phi(\boldsymbol{\pi},\overline{\boldsymbol{\pi}})\,.
\end{aligned}
$$

We can now show that $\hat{\boldsymbol{\pi}}$ is a global minimizer of $\phi(\cdot,\boldsymbol{\pi})$ for any value of $\boldsymbol{\pi}$. We start rewriting $\hat{\boldsymbol{\pi}}$ in terms of ξ_ℓ as follows:

$$
\hat{\pi}_{\ell y} = \frac{1}{Z_\ell} \sum_{(\boldsymbol{x},y')\in\mathcal{T}} \mathbb{1}_{y=y'} \xi_\ell(\boldsymbol{\pi};\boldsymbol{x},y)\,,
$$

where Z_ℓ is the normalizing factor. Then, we have that

$$
\begin{aligned}
&\phi(\hat{\boldsymbol{\pi}},\boldsymbol{\pi}) - \phi(\overline{\boldsymbol{\pi}},\boldsymbol{\pi}) \\
&\quad= -\frac{1}{|\mathcal{T}|} \sum_{(\boldsymbol{x},y)\in\mathcal{T}} \sum_{\ell\in\mathcal{L}} \xi_\ell(\boldsymbol{\pi};\boldsymbol{x},y) \log\left(\frac{\hat{\pi}_{\ell y}}{\overline{\pi}_{\ell y}}\right) \\
&\quad= -\frac{1}{|\mathcal{T}|} \sum_{\ell\in\mathcal{L}} \sum_{y\in\mathcal{Y}} \sum_{(\boldsymbol{x},y')\in\mathcal{T}} \mathbb{1}_{y=y'} \xi_\ell(\boldsymbol{\pi};\boldsymbol{x},y) \log\left(\frac{\hat{\pi}_{\ell y}}{\overline{\pi}_{\ell y}}\right) \\
&\quad= -\frac{1}{|\mathcal{T}|} \sum_{\ell\in\mathcal{L}} \sum_{y\in\mathcal{Y}} Z_\ell \hat{\pi}_{\ell y} \log\left(\frac{\hat{\pi}_{\ell y}}{\overline{\pi}_{\ell y}}\right) \\
&\quad= -\frac{1}{|\mathcal{T}|} \sum_{\ell\in\mathcal{L}} Z_\ell D_{KL}(\hat{\boldsymbol{\pi}}_\ell \| \overline{\boldsymbol{\pi}}_\ell) \leq 0\,,
\end{aligned}
$$

holds for all values of $\overline{\boldsymbol{\pi}}$, where $D_{KL}(\cdot\|\cdot)$ is the Kullback-Leibler divergence. Note that the last inequality yields equality if and only if $\hat{\boldsymbol{\pi}} = \overline{\boldsymbol{\pi}}$, for the Kullback-Leibler divergence yields zero if and only if the two distributions in input coincide. Accordingly, $\hat{\boldsymbol{\pi}}$ is a *strict* global minimizer of $\phi(\cdot,\boldsymbol{\pi})$ for any $\boldsymbol{\pi}$.

As a consequence of the previous derivations we have

$$
R(\Theta,\boldsymbol{\pi};\mathcal{T}) = \phi(\boldsymbol{\pi},\boldsymbol{\pi}) > \phi(\hat{\boldsymbol{\pi}},\boldsymbol{\pi}) \geq R(\Theta,\hat{\boldsymbol{\pi}};\mathcal{T})\,,
$$

where equality holds if and only if we have a fixed point of the update rule (12), *i.e.* if $\hat{\boldsymbol{\pi}} = \boldsymbol{\pi}$. □

References

[1] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, 2014. 1

[2] M. Li, L. Zhou, Z. Yang, A. Li, F. Xia, D. G. Andersen, and A. Smola. Parameter server for distributed machine learning. In *Big Learning NIPS Workshop*, 2013. 1

[3] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 2014. 1

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. 1, 2