A SIMPLE, EFFECTIVE WAY TO IMPROVE NEURAL NET CLASSIFICATION: ENSEMBLING UNIT ACTIVATIONS WITH A SPARSE OBLIQUE DECISION TREE

Arman Zharmagambetov *Miguel Á. Carreira-Perpiñán*

Dept. Computer Science & Engineering, University of California, Merced. Merced, CA, USA
Email: {azharmagambetov,mcarreira-perpinan}@ucmerced.edu

ABSTRACT

We propose a new type of ensemble method that is specially designed for neural nets, and which produces surprising improvements in accuracy at a very small cost, without requiring to train a new neural net. The idea is to concatenate the output activations of internal layers of the neural net into an "ensemble feature vector", and train on this a decision tree to predict the class labels while also doing feature selection. For this to succeed we rely on a recently proposed algorithm to train decision trees - *Tree Alternating Optimization*. This simple procedure consistently improves over simply ensembling the nets in the traditional way, achieving relative error decreases of well over 10% of the original nets on the well known image classification benchmarks. As a subproduct, we also can obtain an architecture consisting of a neural net feature extraction followed by a tree classifier that is faster and more compact than the original net.

***Index Terms*—** image classification, ensemble learning, decision trees, neural networks, feature extraction

1. INTRODUCTION

Deep learning has become highly successful in recent years, in machine learning applications involving complex inputs such as images, audio or text. One reason why this particular type of model works so well is its ability to compute features (activations of neurons in internal layers) that capture important properties of the input (e.g. image), which enable an accurate classification in difficult tasks involving many classes and enormous intra-class variability. These features can also be invariant to certain transformations of the input, such as translation, rotation or intensity changes of an image. As a result, we have now a proliferation of deep net architectures of ever increasing complexity, containing millions of parameters, that continue to improve the state-of-the-art in various tasks. Often these architectures define a "family" of nets of different sizes and correspondingly higher accuracy, such as LeNet [1], VGG [2], ResNet [3], DenseNet [4] and many others. At the same time, the cost of training such models (in computing time, memory size, energy consumption and human expertise required, among other factors) has escalated dramatically, and indeed has motivated much interest in compressing deep nets, from both a research and a practical point of view. In order to achieve state-of-the-art classification in the hardest benchmarks, these nets are trained on large datasets over days of processing in GPU clusters (not including the considerable effort and time dedicated to tune hyperparameters by an expert). This also leads to diminishing returns, as shown in table 1: large increases in size within a family quickly translate into tiny reductions in error.

A different, proven way to construct accurate classifiers is via ensemble learning. In this paper, we propose a new type of ensemble method that is specially designed for neural nets and capitalizes on existing, trained neural nets—a large number of which can be found and online. The idea is to concatenate the output activations of internal layers of the neural net into an *ensemble feature vector*, and train on this a classification tree to predict the class labels. Although using a tree is not strictly necessary, the sparse oblique trees (trained with the recently proposed Tree Alternating Optimization (*TAO*) algorithm [5, 6]) have some unique advantages: they produce trees and forests with high accuracy [7, 8, 9, 10, 11]; they are very fast (at training and inference); and they do feature selection [12]. This is important because the ensemble feature vector can be quite high-dimensional and we expect some features to be redundant. Moreover, preliminary works on combining neural nets and DTs showed promising results [13]. Our simple procedure turns out not to improve significantly if one ensembles features at different layers of the *same* net. But it works surprisingly and consistently well if we ensemble features from *different* neural nets, achieving relative error decreases of well over 10% of the best original net's error. The resulting model strikes a good balance between number of parameters and classification error, improving over the diminishing returns exhibited by existing deep net families.

1.1. Related work

A main focus of research in neural nets, particularly in recent years, has been the development of different architectures, which seek to improve classification accuracy in the first place, but also ease of optimization, and other factors. Convolutional nets [1] are a particularly successful architecture, with feedforward layers constructing progressively more elaborate features of the input. In this context, the idea of feeding intermediate unit activations directly to a classifier layer and training them end-to-end is not new [14, 15]. More recent architectures exploit this idea in various forms (e.g. skip connections): Inception [16], ResNets [3], DenseNet [4], etc. However, in all these works, connecting intermediate neurons directly to a classifier layer is only one part of the definition of the overall neural net architecture, which is a more complex problem requiring careful model selection and training. Here we exploit the potential of taking internal activations from multiple, trained nets, and feeding them to a fast, accurate tree classifier, which is the only part we train.

In ensemble learning, one seeks to train several classifiers and combine them to produce the final classification [17, 18]. Ensembles often show marked improvements over a single classifier, particularly when the ensemble members are diverse. Classic techniques to achieve diversity include training each classifier on a different subset of the training set or having each classifier be of a different type. Classic approaches to combine the classifiers' outputs are voting (e.g. majority or average), boosting, and stacked generalization (SG) [19]. The latter is the most closely related to our work, with a fundamental difference being that in SG one feeds directly

Table 1. Diminishing returns in test classification error (%) vs model size (millions of parameters) in several deep net families: ResNets [3] and VGGs [2] of different numbers of layers on CIFAR-10, and DenseNets [4] of different numbers of layers and blocks on CIFAR-100.

ResNet	size	error	VGG	size	error	DenseNet-BC	size	error
20	0.27M	8.75%	11	9.23M	8.01%	100,12	0.80M	22.27%
32	0.46M	7.51%	13	9.41M	6.48%	94,12	0.97M	22.16%
56	0.85M	6.97%	16	14.72M	6.41%	100,14	1.08M	22.01%
110	1.70M	6.43%	19	20.03M	6.35%	250,24	15.30M	17.60%
1202	19.40M	7.93%				190,40	25.60M	17.18%

the class outputs (e.g. softmax values, posterior probabilities) to the combiner, while we feed internal activations of the neural net, which are generally not directly related to class scores. A further difference is that SG argues for using a validation set (different from the training set used for the learners) whereas our trees are trained on the same training set.

There is also work on training an ensemble of trees on the features produced by a single deep net. Training a forest on the pixel values [20] produces a far lower accuracy than a deep net, but training a forest on the features of a deep net [21] can sometimes (though not always) improve a bit over the deep net accuracy. Some recent works have used features from multiple layers or multiple deep nets to improve classification in the context of a specific application [22, 23, 24]. These works are motivated by sensor fusion in a specific application and use any features available to them in an ad-hoc way. In our work, we study systematically the idea on ensembling deep net features (within- and across nets, and fed to a sparse oblique decision tree) as a generic technique for quick construction of high-accuracy classifiers based on pretrained nets widely available online.

2. THE TAO ALGORITHM

Using the Tree Alternating Optimization (TAO) algorithm is necessary for the success of our approach, because the ensembled features define a high-dimensional vector and we seek a highly accurate sparse decision tree classifier to take advantage of it. This is not possible with traditional trees trained using algorithms such as CART [25] or C4.5 [26]: they have no guarantees concerning the optimization of the desired loss, and the trees are typically axis-aligned (decision node uses a single feature). The *sparse oblique trees* proposed in [5, 6] are trees having linear, multivariate decision nodes but each involving a small number of features, obtained by minimizing an ℓ_1-regularized classification error. Each iteration of TAO is guaranteed to decrease or leave unchanged this objective. Each decision node of the tree uses hard splits, so that an input instance follows a single root-leaf path. This makes the tree inference very fast, unlike with soft decision trees [27], where an instance follows every path and assigns every leaf a certain probability.

We give a very brief description of TAO (see further details in [5]). TAO assumes a given tree structure with initial parameter values, and minimizes the following objective function jointly over the parameters $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_i\}$ of all nodes i of the tree:

$$E(\boldsymbol{\Theta}) = \sum_{n=1}^{N} L(y_n, T(\mathbf{x}_n; \boldsymbol{\Theta})) + \lambda \sum_{\text{nodes } i} \|\mathbf{w}_i\|_1 \quad (1)$$

where $\{(\mathbf{x}_n, y_n)\}_{n=1}^{N} \subset \mathbb{R}^D \times \{1, \dots, K\}$ is a training set of D-dimensional real-valued instances and their labels (in K classes), $L(\cdot, \cdot)$ is the 0/1 loss, and $T(\mathbf{x}; \boldsymbol{\Theta})$: $\mathbb{R}^D \rightarrow \{1, \dots, K\}$ is the predictive function of the tree. The parameters $\boldsymbol{\theta}_i$ at a node i are a label if i is a leaf[1] or a hyperplane with weight vector $\mathbf{w}_i \in \mathbb{R}^D$ and bias $b_i \in \mathbb{R}$ if i is a decision node, which thus sends an input instance $\mathbf{x}$ down its right child if $\mathbf{w}_i^T \mathbf{x} \geq b_i$ and down its left child otherwise.

We take as initial tree a deep enough, complete binary tree with random parameters. In order to train such trees, TAO relies on two theorems. First, a separability condition which states that $E(\boldsymbol{\Theta})$ in eq. (1) separates over nodes that are not descendants of each other (e.g. all nodes at the same depth). Second, the problem of optimizing eq. (1) over a node is equivalent to a much simpler "reduced problem". This consists of optimizing a linear binary classifier over $\{\mathbf{w}_i, b_i\}$ on the training instances $\{(\mathbf{x}_n, \overline{y}_n)\}$ that currently reach node i. Each such instance $\mathbf{x}_n$ is assigned a pseudolabel $\overline{y}_n \in \{\texttt{left}, \texttt{right}\}$ which indicates the child whose subtree gives the better prediction for $\mathbf{x}_n$. Hence, TAO repeatedly trains an ℓ_1-regularized binary classifier at the decision nodes and a K-class classifier at the leaves (cycling over all nodes depthwise).

3. DEEP NET FEATURE ENSEMBLING WITH A SPARSE OBLIQUE TREE CLASSIFIER

Our proposed procedure is as follows. Assume we have a dataset of input instances and their labels (in K classes); and several **deep nets** trained, somehow, on that dataset. Each deep net has a number of layers, each layer has a number of units (neurons), and each unit computes one **feature**. This includes the input layer (pixel values, for an image) and the output layer (softmax outputs). Then we first construct an **ensemble feature vector** by picking a subset of features from each net and concatenating them; and then we train a **sparse oblique tree classifier** with TAO on a dataset where the inputs are the ensemble feature vectors and the output is their ground-truth label. This procedure is generic and admits multiple variations. Some important ones are:

Features We may have just one deep net and ensemble features from its internal layers (called *within net* in our experiments); or multiple nets of different types and pick features from a different layer in each (we call this *across nets*).

Deep net training We may train the individual nets ourselves (using standard ensemble learning techniques to generate diversity, such as training each net on a different sample). But practically it may be better to take existing deep nets that may have been trained in (partially) unknown ways. The latter may seem risky but it is much simpler, it introduces diversity, and the tree can select which features to use anyway (by selecting the sparsity hyperparameter λ).

Tree training We train a tree using TAO algorithm as it was mentioned above. We tune the sparsity hyperparameter λ to achieve a desirable trade-off between validation error and sparsity in the tree. Usually, one will want to pick the sparsest tree whose validation error is reasonably close to minimum validation error (see [28, fig. 7.9]).

Our experiments explore several of these variations. We also compare with standard neural net ensembles and stacked generalization.

[1] We extend TAO to learn a softmax classifier at each leaf by using the cross-entropy instead of the 0/1 loss in eq. (1).

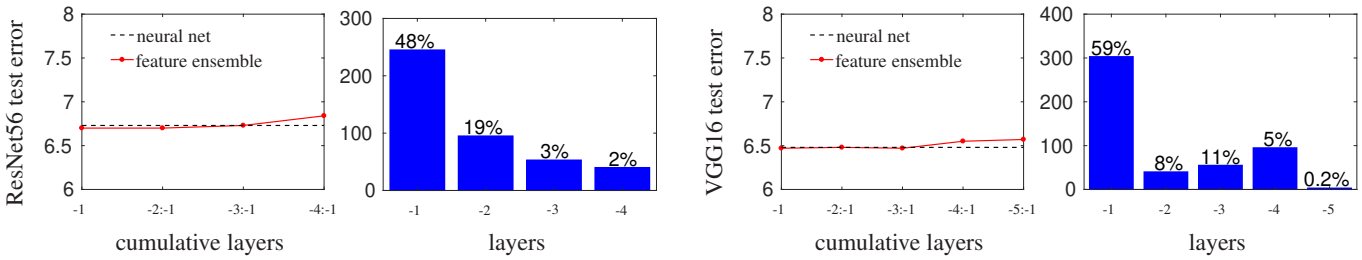

Fig. 1. Ensembling features within the same net: ResNet56 (left) and VGG16 (right) on CIFAR-10. In each panel, we show the test error and histogram of features selected per layer.

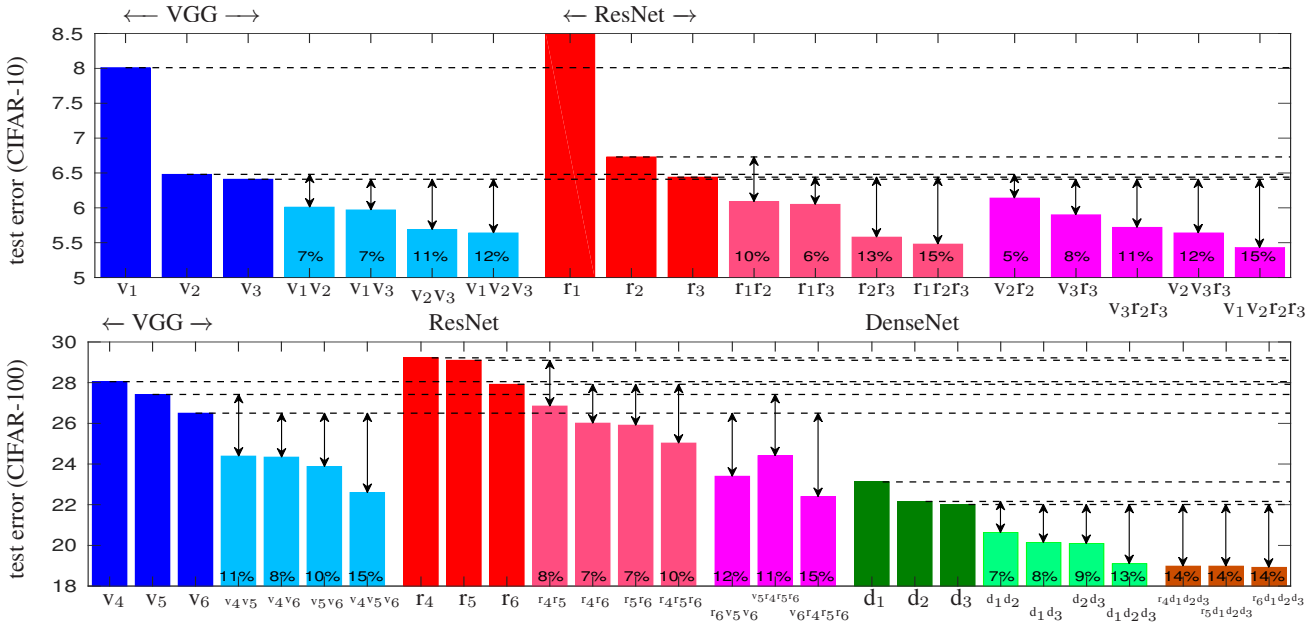

Fig. 2. Ensembling features across different nets (top: CIFAR-10, bottom: CIFAR-100). The horizontal dashed lines and vertical arrows indicate the improvement of a feature ensemble over the best member of the ensemble (also given as relative error in percentage inside each bar).

4. EXPERIMENTS

We consider three families of deep nets: VGG [2], ResNet [3] and DenseNet [4], each with several sizes (number of layers for VGG and ResNet, number of layers and blocks for DenseNet). For CIFAR-10 we train nets ourselves; for CIFAR-100 we use available trained nets from the web[2].

TAO uses an ℓ_1-regularized logistic regression to solve the decision node optimization (using LIBLINEAR [29]). The complexity of one TAO iteration (pass over all nodes) is $\mathcal{O}(\Delta C)$ where Δ is the tree depth (for a complete tree) and C the cost of one ℓ_1-regularized logistic regression on the entire training set. Training a tree for the largest experiments took at most 18 minutes, with a final depth of the resulting, pruned tree of 2 to 3 typically. *The inference time* and *memory size* are dominated by the computation of the deep net(s) features; the tree accounts for just 0.1% and 1.3% of the time and memory.

Summary. Our experiments show the following: within-net feature ensembling does not noticeably improve accuracy, but across-net (same or different architectures) improves it considerably. In the latter case, our feature ensembling consistently improves over the standard neural net ensembling.

4.1. Within net

Modern deep nets can have many layers and a huge number of neurons, hence potential features to ensemble in a vector. We cannot carry out a comprehensive evaluation of all possible combinations, so we provide results in a subset of what we thought might be interesting combinations instead. Firstly, we include entire layers at once or not at all. Second, we do not use the very last layer (softmax outputs) because it can always be computed by the tree (by a softmax classifier at each leaf). Third, we report a subset of cumulative ensembles containing the features from layer i to the penultimate layer (before the softmax outputs), which we notate as "$-i{:}-1$" (so "$-2{:}-1$" contains the features from the last two layers). Fig. 1 shows our results for ResNet56 and VGG16 on CIFAR-10.

On both deep nets, which have rather different architectures, we clearly observe that, as we add intermediate features to the penultimate-layer features, the training error decreases monotonically but very little, and the test error decreases very little then increases slightly (likely because the tree is overfitting). Inspecting

[2] https://github.com/bearpaw/pytorch-classification

the features used in the tree (histogram plot) shows that indeed most features come from the penultimate layer. Adding the raw pixel inputs did not help either. Training a separate tree on each layer's features and ensembling the trees by voting (results not shown) achieves very similar results. This is a negative result, but interesting regarding the behavior of a deep net: it suggests that the features in the penultimate layer are sufficient for optimal classification, and that the features in previous layers are correlated or redundant with them (although still necessary to construct them).

4.2. Across nets

Fig. 2 shows our results for ResNet, VGG and DenseNet (of different sizes) on CIFAR-10 and CIFAR-100. Based on the results of the within-net experiments, for each net we use only the features in the penultimate layer (before the softmax outputs). We notate each deep net by an id and each combination by the ids making it up. For CIFAR-10: v_1, v_2, v_3 stand for VGG11 / 13 / 16; and r_1, r_2, r_3 for ResNet20 / 56 / 110, respectively. For CIFAR-100: v_4, v_5, v_6 stand for VGG19 / 16 / 13; r_4, r_5, r_6 for ResNet56 / 80 / 110; and d_1, d_2, d_3 for DenseNet-BC-100(k=12) / 94(k=14) / 100(k=14), respectively. Therefore $v_3r_2r_3$ means concatenating features from VGG16, ResNet56 and ResNet110 on CIFAR-10.

We clearly see that any combination improves over all its members. The improvement is particularly remarkable if we bear in mind the diminishing returns of table 1; It is also remarkable in that we achieve large gains with few members (2 to 4) in the ensemble: the relative improvement of the combination over the best member net is between 5% and 15%. This holds when combining nets of different size, architecture, or both. The more members (of different type), the lower the error (with eventually diminishing returns); but other than that there does not seem to be a clear pattern of what combination is better. Presumably the features from the different nets are less correlated so their concatenation can benefit.

4.3. Comparison with baselines

An even simpler and well-known way to combine multiple existing nets is by neural net ensembling, where, given the softmax outputs for each class of each net, we either pick the label by majority vote (assuming an odd number of members) or sum the softmax outputs and pick the label with largest sum. Our results in table 2 (similar results were obtained for ResNets) show that this does improve over the individual members as well, but not as much as our approach, which reduces the error of neural net ensembling by up to 5.3%. While our approach has an extra overhead (learning the tree), this is minor at training time and negligible at inference time. We also compared our approach against traditional stacked generalization (see NN stacking in table 2). We ensembled the softmax outputs of the neural nets (not the deeper features) into a super-feature vector which we fed to a logistic regression. This approach improves over the individual neural nets as well, but less than our method, which beats it by up to 4%.

Finally, we also compare with more advanced baselines in table 3. Namely, [30] employs a neural net architecture with several parallel branches and the output of all branches are averaged to produce the final result. In SGDR-WRN [31], authors suggest to take snapshots of several SGD restarts and ensemble them. These methods train ensemble of NNs in end-to-end manner, whereas our approach builds ensemble based only on features. But it is remarkable that our simple approach outperforms the mentioned works. For all methods, we compare results of neural nets of the similar sizes (# of parameters) and report test error.

Table 2. Comparison of TAO trees with traditional neural net ensembling and stacked generalization for VGGs on CIFAR-10. The ensemble of neural nets is done using majority vote ("NN ensemble (m.v.)") or computing the average (per class) of the softmax outputs and picking the largest ("NN ensemble (a.s.)"). "NN stacking" uses logistic regression trained on concatenated outputs of the neural net.

Method	E_{train} (%)	E_{test} (%)
v_1	0.0	8.01
v_2	0.0	6.48
v_3	0.0	6.41
v_1v_2 - TAO tree	0.0	**5.98**
v_1v_2 - NN ensemble (a.s.)	0.0	6.23
v_1v_2 - NN ensemble (m.v.)	-	-
v_1v_2 - NN stacking	0.0	6.11
v_1v_3 - TAO tree	0.0	**5.95**
v_1v_3 - NN ensemble (a.s.)	0.0	6.13
v_1v_3 - NN ensemble (m.v.)	-	-
v_1v_3 - NN stacking	0.0	6.16
v_2v_3 - TAO tree	0.0	**5.63**
v_2v_3 - NN ensemble (a.s.)	0.0	5.73
v_2v_3 - NN ensemble (m.v.)	-	-
v_2v_3 - NN stacking	0.0	5.70
$v_1v_2v_3$ - TAO tree	0	**5.60**
$v_1v_2v_3$ - NN ensemble (m.v.)	0.0	5.9
$v_1v_2v_3$ - NN ensemble (a.s.)	0.0	5.65
$v_1v_2v_3$ - NN stacking	0.0	5.68

Table 3. Comparison of our approach with some baselines on neural net ensembling and single DenseNet on CIFAR-100. "*" indicates that the result is taken from [30].

Method	E_{test} (%)	# params
DenseNet-BC(100,14)	22.01	1.1M
SGDR-WRN*	20.90	2.4M
Coupled NN*	19.95	2.3M
TAO($d_1d_2d_3$)	**19.07**	2.8M

5. CONCLUSION

We have proposed a new form of ensembles specifically tailored for deep nets, based on the hypothesis that internal features rather than class scores contain information that helps classification, and that this can be captured by a sparse oblique tree. While this barely improves within a single net, it significantly and consistently improves if ensembling features across different nets. The decision tree, trained with the TAO algorithm, allows us to handle efficiently and accurately the resulting high-dimensional feature vector. Inspecting the tree confirms that features in early layers of the net do not help improve classification.

This provides a simple, fast and practical way to construct state-of-the-art classifiers, by downloading existing deep nets and training the tree (in a few minutes' runtime). Its accuracy consistently beats ensembling the nets via voting. Our results are remarkably robust across a range of deep nets which were trained for the image classification task. Finally, our ideas suggest future work in jointly training the entire architecture, using a forest classifier (rather than a tree), and in neural net compression.

Acknowledgments. Work partially supported by NSF award IIS–2007147.

6. REFERENCES

[1] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[2] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR'15)*, San Diego, CA, May 7–9 2015.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *CVPR'16*, Las Vegas, NV, June 26 – July 1 2016, pp. 770–778.

[4] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger, "Densely connected convolutional networks," in *CVPR'17*, Honolulu, HI, July 21–26 2017, pp. 2261–2269.

[5] Miguel Á. Carreira-Perpiñán and Pooya Tavallali, "Alternating optimization of decision trees, with application to learning sparse oblique trees," in *NeurIPS'18*.

[6] Miguel Á. Carreira-Perpiñán, "The Tree Alternating Optimization (TAO) algorithm: A new way to learn decision trees and tree-based models," arXiv, 2021.

[7] Arman Zharmagambetov, Suryabhan Singh Hada, Magzhan Gabidolla, and Miguel Á. Carreira-Perpiñán, "Non-greedy algorithms for decision tree optimization: An experimental comparison," in *Int. J. Conf. Neural Networks (IJCNN'21)*, Virtual event, July 18–22 2021.

[8] Arman Zharmagambetov and Miguel Á. Carreira-Perpiñán, "Smaller, more accurate regression forests using tree alternating optimization," in *ICML'20*, Online, July 13–18 2020, pp. 11398–11408.

[9] Miguel Á. Carreira-Perpiñán and Arman Zharmagambetov, "Ensembles of bagged TAO trees consistently improve over random forests, AdaBoost and gradient boosting," in *FODS'20*, Seattle, WA, Oct. 19–20 2020, pp. 35–46.

[10] Arman Zharmagambetov, Magzhan Gabidolla, and Miguel Á. Carreira-Perpiñán, "Improved boosted regression forests through non-greedy tree optimization," in *Int. J. Conf. Neural Networks (IJCNN'21)*, Virtual event, July 18–22 2021.

[11] Arman Zharmagambetov, Magzhan Gabidolla, and Miguel Á. Carreira-Perpiñán, "Improved multiclass AdaBoost for image classification: The role of tree optimization," in *IEEE Int. Conf. Image Processing (ICIP 2021)*, Anchorage, AK, Sept. 19–22 2021.

[12] Suryabhan Singh Hada, Miguel Á. Carreira-Perpiñán, and Arman Zharmagambetov, "Sparse oblique decision trees: A tool to understand and manipulate neural net features," arXiv:2104.02922, Apr. 7 2021.

[13] Arman Zharmagambetov and Miguel Á. Carreira-Perpiñán, "Learning a tree of neural nets," in *Proc. of the IEEE Int. Conf. Acoustics, Speech and Sig. Proc. (ICASSP'21)*, Toronto, Canada, June 6–11 2021, pp. 3140–3144.

[14] Pierre Sermanet, , and Yann LeCun, "Traffic sign recognition with multi-scale convolutional networks," in *Int. J. Conf. Neural Networks (IJCNN'11)*, San Jose, CA, July 31 – Aug. 5 2011, pp. 2809–2813.

[15] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu, "Deeply-supervised nets," in *Proc. of the 18th Int. Conf. Artificial Intelligence and Statistics (AISTATS 2015)*, Guy Lebanon and S. V. N. Vishwanathan, Eds., San Diego, CA, May 10–12 2015, pp. 562–570.

[16] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich, "Going deeper with convolutions," in *CVPR'15*, Boston, MA, June 7–12 2015, pp. 1–9.

[17] Thomas G. Dietterich, "Ensemble methods in machine learning," in *Multiple Classifier Systems*, pp. 1–15. Springer-Verlag, 2000.

[18] Ludmila I. Kuncheva, *Combining Pattern Classifiers: Methods and Algorithms*, John Wiley & Sons, second edition, 2014.

[19] David H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, no. 2, pp. 241–259, 1992.

[20] Samuel Schulter, Paul Wohlhart, Christian Leistner, Amir Saffari, Peter M. Roth, and Horst Bischof, "Alternating decision forests," in *CVPR'13*, Portland, OR, June 23–28 2013, pp. 508–515.

[21] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Buló, "Deep neural decision forests," in *Proc. 15th Int. Conf. Computer Vision (ICCV'15)*, Santiago, Chile, Dec. 11–18 2015, pp. 1467–1475.

[22] L. D. Nguyen, D. Lin, Z. Lin, and J. Cao, "Deep CNNs for microscopic image classification by exploiting transfer learning and feature concatenation," in *IEEE International Symposium on Circuits and Systems (ISCAS 2018)*, May 2018, pp. 1–5.

[23] Max Schwarz, Hannes Schulz, and Sven Behnke, "RGB-D object recognition and pose estimation based on pre-trained convolutional neural network features," in *Proc. of the 2015 IEEE Int. Conf. Robotics and Automation (ICRA'15)*, Seattle, Washington, May 26–30 2015, pp. 1329–1335.

[24] U. K. Lopes and J. F. Valiati, "Pre-trained convolutional neural networks as feature extractors for tuberculosis detection," *Computers in Biology and Medicine*, vol. 89, pp. 135–143, Oct. 1 2017.

[25] Leo J. Breiman, Jerome H. Friedman, R. A. Olshen, and Charles J. Stone, *Classification and Regression Trees*, Wadsworth, Belmont, Calif., 1984.

[26] J. Ross Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.

[27] Michael I. Jordan and Robert A. Jacobs, "Hierarchical mixtures of experts and the EM algorithm," *Neural Computation*, vol. 6, no. 2, pp. 181–214, Mar. 1994.

[28] Trevor J. Hastie, Robert J. Tibshirani, and Jerome H. Friedman, *The Elements of Statistical Learning—Data Mining, Inference and Prediction*, Springer Series in Statistics. Springer-Verlag, second edition, 2009.

[29] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin, "LIBLINEAR: A library for large linear classification," *J. Machine Learning Research*, vol. 9, pp. 1871–1874, Aug. 2008.

[30] Anuvabh Dutt, Denis Pellerin, and Georges Quénot, "Coupled ensembles of neural networks," *Neurocomputing*, vol. 396, pp. 346–357, July 5 2020.

[31] Ilya Loshchilov and Frank Hutter, "SGDR: Stochastic gradient descent with restarts," in *ICLR'17*, Toulon, France, Apr. 24–26 2017.