Neural Forest Learning

Yun-Hao Cao and Jianxin Wu
National Key Laboratory for Novel Software Technology
Nanjing University, Nanjing, China
{caoyunhao1997, wujx2001}@gmail.com

Abstract

We propose Neural Forest Learning (NFL), a novel deep learning based random-forest-like method. In contrast to previous forest methods, NFL enjoys the benefits of end-to-end, data-driven representation learning, as well as pervasive support from deep learning software and hardware platforms, hence achieving faster inference speed and higher accuracy than previous forest methods. Furthermore, NFL learns non-linear feature representations in CNNs more efficiently than previous higher-order pooling methods, producing good results with negligible increase in parameters, floating point operations (FLOPs) and real running time. We achieve superior performance on 7 machine learning datasets when compared to random forests and GBDTs. On the fine-grained benchmarks CUB-200-2011, FGVC-aircraft and Stanford Cars, we achieve over 5.7%, 6.9% *and* 7.8% *gains for VGG-16, respectively. Moreover, NFL can converge in much fewer epochs, further accelerating network training. On the large-scale ImageNet ILSVRC-12 validation set, integration of NFL into ResNet-18 achieves top-1/top-5 errors of* $28.32\%/9.77\%$*, which outperforms ResNet-18 by* $1.92\%/1.15\%$ *with negligible extra cost and the improvement is consistent under various architectures.*

1. Introduction

Deep convolutional neural networks (CNNs) have achieved remarkable advancement in a variety of computer vision tasks, such as image classification ([18, 28, 9]), object detection ([6, 8]) and semantic segmentation ([22]). Despite the rapid development of CNNs, forest methods such as random forests [1] or gradient boosting trees (GBDTs) [5] are still the dominant way for vectorized inputs and and are widely used in many real-world applications [31]. Training and predictions for these models are computationally expensive for large problems. More importantly, such forest methods are mostly combinatorial rather than differentiable and they lack the capability of representation learning. In contrast, CNNs integrate representation learning and classifier learning in an end-to-end fashion, enjoying pervasive software (e.g., deep learning frameworks) and hardware (e.g., GPUs) support. Hence, one question arises: Can we design a forest method with both the capability of end-to-end representation learning and the support of existing software and hardware for deep learning?

An opposite aspect is to examine CNNs. By stacking layers of convolution and nonlinearity, CNNs effectively learn from low-level to high-level features and discriminative representations. As one standard module in deep CNN architectures, global average pooling (GAP) summarizes linear statistics of the last convolution layer. Recently, many higher-order pooling (HOP) methods (e.g., [21]) are proposed to be integrated into deep CNNs to learn higher-order, non-linear feature representations to replace GAP, which have achieved impressive recognition accuracies. However, these HOP methods suffer from expensive computing costs because of the need to calculate covariance information of very high dimensional matrices. Therefore, another question is: Can we add non-linearity to the linear GAP to achieve both good accuracy and high efficiency?

In this paper, we take a step towards addressing these two problems jointly. We propose a Neural Forest Learning (NFL) model which is a deep learning based random-forest-like method. On one hand, it learns non-linear decision tree representations using both randomness and existing CNN layers, which enjoys the benefits of end-to-end, data-driven representation learning, as well as pervasive support from deep learning software and hardware platforms. Moreover, NFL handles vectorized inputs well and achieves both higher accuracy and faster inference speed than random forests and GBDTs, which are attractive for many real-world pattern recognition tasks.

On the other hand, NFL can be seamlessly installed after the convolution layers and a GAP layer at the end of a CNN for image recognition, which non-linearly transforms the output of GAP. NFL achieves higher accuracy than standard GAP and is more efficient than HOP methods, with negligible increase in parameters, FLOPs and real running time. Furthermore, NFL can be installed across all layers in

a CNN when integrated into Squeeze-and-Excitation (SE) networks [12] and it achieves comparable or better accuracy than SENet with fewer parameters and FLOPs.

Experimental results confirm the effectiveness of NFL. We achieve superior performance on 7 machine learning datasets when compared to random forests and GBDTs. On the fine-grained benchmarks CUB-200-2011 [33], FGVC-aircraft [24] and Stanford Cars [16], by combining NFL we achieve over 5.7%, 6.9% and 7.8% gains for VGG-16, respectively, with negligible increase in parameters, FLOPs and real running time. On ImageNet ILSVRC-12 [26], integration of NFL into ResNet-18 achieves top-1/top-5 errors of $28.32\%/9.77\%$, which outperforms ResNet-18 by $1.92\%/1.15\%$ with negligible extra cost.

2. Related Work

2.1. Forest Learning

Forest learning is a powerful learning paradigm which often uses decision trees as its base learners. Bagging and boosting, for instance, are the driving forces of random forests [1] and GBDTs [5], respectively. They have become the choice for many industrial applications and data science projects, ranging from predicting clicks on Ads [10] to numerous data science competitions in Kaggle[1] and beyond. Note that the input to such models are vectors rather than images, and not suitable for methods such as CNN to process. Very recently, ThunderGBM [34] proposes a GPU-based software to improve the efficiency of random forests and GBDTs, especially for large and high dimensional problems. Nevertheless, they are designed for specific algorithms and hardwares. With the rapid development of deep learning, there have also been deep forest methods. Recently, [35] proposes gcForest, which is a deep forest ensemble with a cascade structure. mGBDTs [4] learns hierarchical distributed representations by stacking several layers of regression GBDTs as its building block. In contrast, our method integrates forest with end-to-end, data-driven representation learning capabilities under the support of existing deep learning software and hardware platforms. NDF [15] combines a single deep CNN with a random forest for image classification, where outputs of the top CNN layer are considered as nodes of the decision tree and prediction loss is computed at each split node of the tree. Despite sharing the similarity that both of us are bagging methods based on deep learning, our work differs as follows: (i) We implement the bagging process in a novel and easy way, which will be introduced in the next section. (ii) Our method is light-weight and more easily integrated into existing deep learning frameworks.

[1]https://www.kaggle.com

2.2. Non-linear representations in CNNs

Statistics higher than first-order ones have been successfully used in both classic and deep learning based classification scenarios. The Vectors of Locally Aggregated Descriptors (VLAD) [14] and Fisher Vectors (FV) [25] use non-linear representations based on hand-crafted features (e.g., SIFT [23]). By replacing hand-crafted features by features extracted from CNNs pretrained on ImageNet [26], these models achieve state-of-the-art results on many recognition tasks [2]. In these designs, image representation and classifier training are not jointly optimized and end-to-end training has not been fully studied. [21] proposes a bilinear CNN (B-CNN) that aggregates the outer products of convolutional features from two networks and allows end-to-end training for fine-grained visual classification. [20] proposes an iterative matrix square root normalization (iSQRT) method for fast training of global covariance pooling networks. These works have shown that higher-order, non-linear feature representations based on convolution outcomes achieve impressive improvements over the classic linear representations. However, they suffer from the expensive computational overhead because these methods depend heavily on eigendecomposition or singular value decomposition of very high dimensional covariance matrices. Contrary to previous higher-order methods, our NFL learns non-linear feature representations with negligible increase in parameters, FLOPs and real running time while achieving higher accuracy.

3. Neural Forest Learning

In this section, we propose the NFL model, which mainly consists of random permutations and group convolutions. Furthermore, we show that it in effect is an ensemble of one-level trees, hence is named the Neural Forest Learning (NFL).

3.1. Network architecture

The following notations are used in the rest of this paper. Here we use $\boldsymbol{x} \in \mathbb{R}^d$ to represent a d-dimensional feature vector and x_i to represent the i-th feature ($i = 1, \ldots, d$) of $\boldsymbol{x}$. We denote the expansion rate in depth as $nMul$, expansion height as dH, expansion width as dW and the number of channels per group in a group convolution as $nPer$.

Our goal is to build a neural forest which combines the advantages of both forest learning and deep learning. We propose a novel NFL architecture to achieve this goal, as shown in Figure 1. For a d-dimensional feature vector $\boldsymbol{x}$, we first generate m random permutations $\boldsymbol{\sigma}^1, \ldots, \boldsymbol{\sigma}^m$, where $m = dH \times dW \times nMul$. Then we can obtain a set of randomly permuted vectors $Z = \{\boldsymbol{z}^t\}$ from $\boldsymbol{x}$ correspondingly:

$$\boldsymbol{z}^t = (x_{\boldsymbol{\sigma}^t_1}, \ldots, x_{\boldsymbol{\sigma}^t_d}), \quad t = 1, \ldots, m\,. \tag{1}$$

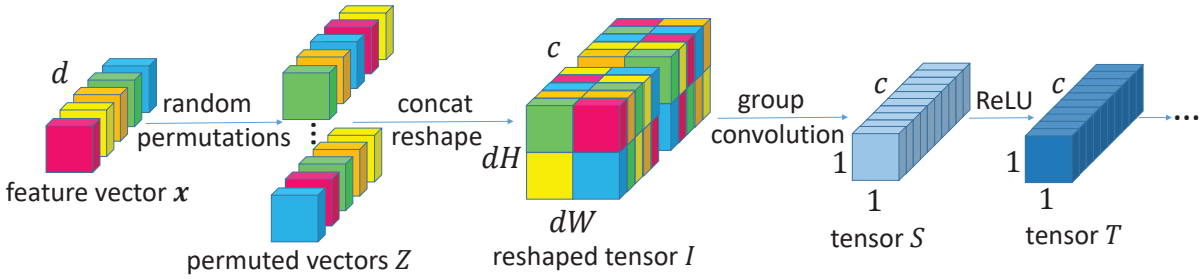


Figure 1: NFL architecture with one group convolution layer.

Then, we concatenate these d-dimensional feature vectors into one vector U of $m \times d$ dimensions and reshape U into an order-3 tensor $I \in \mathbb{R}^{dH \times dW \times c}$, where $c = nMul \times d$. We denote the element at location of i-th row, j-th column and k-th channel in I as $I(i,j,k)$. $I(i,j,k)$ is actually generated by t-th random permutation $\boldsymbol{\sigma}^t$ and we have:

$$t = \lfloor k/d \rfloor \times dH \times dW + (i-1) \times dW + j\,. \quad (2)$$

Hence, $I(i,j,k)$ corresponds to s-th element in $\boldsymbol{z}^t$ and we have:

$$I(i,j,k) = x_{\boldsymbol{\sigma}^t_s}\,, \quad (3)$$

where $s = k \mod d$, and t is calculated by Equation (2). Then, we send the tensor I into a group convolution layer of kernel size (kH, kW), out channel numbers c and group numbers $\lfloor c/nPer \rfloor$ without padding, obtaining a new order-3 tensor S of size (oH, oW, c) followed by ReLU non-linearity. We can directly use only one group convolution layer with $kH = dH$ and $kW = dW$, thus achieving S of size $(1,1,c)$, as is done in Figure 1. Also, we can use multiple group convolution layers with $kH < dH$ and $kW < dW$ to make our forest deeper. Then, we add ReLU non-linearity upon S and obtain tensor T of size $(1,1,c)$. Finally, we send T into fully connected layers plus a softmax layer for classification.

3.2. Neural forest learning via CNN implementation

For a d-dimensional input feature vector, it can be either a hand-crafted feature vector in traditional machine learning or pattern recognition tasks or a feature representation generated by CNNs (e.g., the output of a GAP layer). It is then transformed into the tensor I by random permutations.

I includes a set of 2-D feature maps $I = \{I^k\}(k = 1, \ldots, c)$. I^k, of size $dH \times dW$, is the k-th feature map of the corresponding channel (the k-th channel). For each feature map I^k, it consists of $dH \times dW$ features, which are randomly sampled from original features. Then, each group convolution filter which randomly chooses $kH \times kW \times nPer$ features and the subsequent ReLU layer which acts upon an attribute (aka linear combination of these selected features) can be considered as a one-level decision tree. We

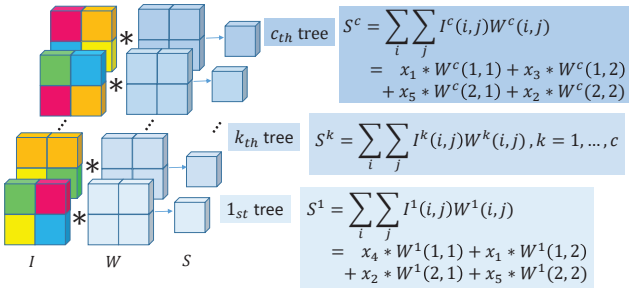


Figure 2: Group convolution makes a forest. Each square in different color corresponds to different feature in input feature vector $\boldsymbol{x}$ of 5 dimensions in Figure 1, e.g., the red square corresponds to x_1, etc.

take a group convolution layer with $nPer = 1$ (aka depthwise convolution) as an example to illustrate this relationship, which is shown in Figure 2. We use W^k to denote the k-th depthwise convolution filter, S^k to denote the k-th channel of S and T^k to denote the k-th channel of T $(k = 1, \ldots, c)$. Then from Equation (3) we have:

$$\begin{aligned} S^k &= \sum_i \sum_j I^k(i,j) W^k(i,j) \\ &= \sum_i \sum_j I(i,j,k) \times W(i,j,k) \\ &= \sum_i \sum_j x_{\boldsymbol{\sigma}^t_s} \times W(i,j,k)\,. \end{aligned} \quad (4)$$

Let $f(\cdot)$ denote the ReLU function, T^k can be computed as:

$$T^k = f(S^k) = \begin{cases} S^k & S^k \geq 0 \\ 0 & S^k < 0 \end{cases} \quad (5)$$

Then, from Equation (5) and Figure 2 we can see that each convolution filter W^k plus the subsequent ReLU corresponds to a tree which outputs a linear combination of features and decision based on it. Hence, all convolution filters form a forest with c different one-level trees. Actually, if we use 1×1 depthwise group convolution where $kH = kW = 1$, each base learner reduces to a decision

stump which learns with a single feature. In conclusion, random permutation operation acts as resampling and group convolution is used for aggregation. These operations in effect form a feature bagging process where each base learner learns from a random subset of input features.

Although NFL is not strictly a forest because *it does not match traditional trees or forest models precisely*, we call it neural forest learning because NFL and forest models are similar in their motivations.

Also, when we increase $nMul$, the number of channels c gets larger and we get more group convolution filters. Hence, from Equation (4), more trees are integrated and our forest get larger correspondingly. Furthermore, we can increase $nPer$ and dH/dW to explicitly increase the number of features utilized in each tree, thus increasing the capacity of each base learner. Finally, by stacking more group convolution layers, we can make our forest deeper. In all our experiments, we set $dH = dW$ for simplicity, so we denote dH and dW as dH/dW in the rest of this paper. We conduct ablation studies about $nMul, nPer$ and dH/dW in Sec 4.3.

4. Experimental Results

We will empirically study the benefits of our NFL method. On one hand, for vectorized inputs, we compare our method with other competitive forest methods on 6 machine learning classification datasets: satimage [11], gisette [7], mnist [19], letter [11], usps [13] and yeast [3] as well as 1 multivariate regression dataset sarcos [32]. On the other hand, NFL can be integrated into CNNs for improving non-linear capability either at the end of the network or across all layers in the network. We conduct experiments on CIFAR-10 [17], CIFAR-100 [17], fine-grained visual categorization benchmarks and large-scale ImageNet ILSVRC-12 [26] task with five widely used deep models: MobileNetV2 [27], VGG [28], ResNet [9], Inception-v3 [29] and SENet [12]. All our experiments were conducted using PyTorch on Tesla M40 GPUs.

4.1. Datasets

For machine learning classification and regression datasets, a brief description of them including the train-test split, the number of categories and feature dimensions is given in the appendix. The CIFAR-10 [17] consists of 50000 training images and 10000 test images in 10 classes and the CIFAR-100 [17] is just like the CIFAR-10, except it has 100 classes containing 600 images each. For fine-grained categorization, we use three popular fine-grained benchmarks, i.e., CUB-200-2011 (Birds) [33], FGVC-aircraft (Aircrafts) [24] and Stanford Cars (Cars) [16]. The Birds dataset contains 11788 images from 200 species, with large intra-class variation but small inter-class variation. The Aircrafts dataset includes 100 aircraft classes and a total of 10000 images with small background noise but higher inter-class similarity. The Cars dataset consists of 16185 images from 196 classes. For large-scale image classification, we adopt the ImageNet ILSVRC-12 dataset [26] with 1000 object categories. The dataset contains 1.28M images for training, 50K images for validation and 100K images for testing (without published labels). As in [9], we report the results on the validation set.

4.2. Machine learning datasets

We compare NFL with forest methods, e.g., random forests and GBDTs in terms of accuracy, training/testing time and model size. Furthermore, because we use NFL with 2 fully connected (fc) layers on these machine learning classification and regression datasets, we also compare it with multi-layer perceptrons (MLP).

Implementation details: We build NFL by 1 group convolution layer and 2 fc layers with batch normalization in all datasets. We construct MLP with 2 fc layers with batch normalization in the same way. We split 10% of the training data for validation to determine the total epochs separately for each dataset. We train all networks for 20˜50 epochs, using Adam as optimizer and initializing learning rate to 0.0001. In Table 1, We set $nPer$ to 1 and dH/dW to 3 for all these datasets for simplicity. Considering feature dimensions among different datasets, we set $nMul$ to 20, 10, 16, 100, 30, 20 and 40 for satimage, gisette, mnist, letter, usps, yeast and sarcos, respectively. For MLP, random forests and GBDTs, we carefully tune the parameters through 5-fold cross-validation on the train set and choose the best parameters for them in each dataset. We report the mean accuracy and standard deviation of 5 trails for all datasets except yeast, which is evaluated by 10-fold cross-validation. We use different experimental settings for algorithms in Table 2 for fair model size, speed and accuracy comparison and the experimental details are included in the appendix.

Comparison among different algorithms: Table 1 shows that NFL achieves the highest accuracy in all classification datasets and the lowest mean square error (MSE) in the regression dataset compared with MLP, random forests, GBDTs, NDF [15] and ANT [30]. Moreover, it is worth noting that although our method introduces randomness due to random permutations, it achieves a low standard deviation and is very robust, even more stable than MLP.

Using the 2 *largest* datasets and the *highest* dimensional dataset among these 6 classification datasets, we compare the speed and size of NFL with random forests and GBDTs. Table 2 shows that although GPU-based ThunderGBM can greatly reduce the training time, especially for GBDTs, the inference process seems to have no benefit. Compared to these forest methods, NFL achieves the highest accuracy and the fastest inference speed on mnist and letter, and also the smallest model size on letter. NFL achieves the highest

Table 1: Accuracy(%) on 6 machine learning classification datasets and MSE on the regression dataset sarcos. We report the average accuracy and standard deviation of 5 trails. NFL and MLP are the results of last epoch. * denotes that [30] didn't report the standard deviation.

Datasets	NFL (ours)	MLP	NDF [15]	Random Forests	GBDTs
satimage	**91.52±0.31**	90.01±0.31	89.71±0.31	91.01±0.35	89.26±0.04
gisette	**98.26±0.05**	98.08±0.12	97.24±0.29	96.98±0.13	97.18±0.04
mnist	**98.57±0.03**	98.17±0.07	97.29±0.12	96.96±0.08	96.56±0.07
letter	**97.85±0.10**	97.23±0.17	97.08±0.17	96.14±0.10	94.66±0.01
usps	**95.71±0.17**	95.13±0.26	94.99±0.24	93.80±0.19	92.83±0.03
yeast	**62.81±2.61**	60.57±3.45	60.31±3.37	62.81±3.47	60.71±2.35
	NFL (ours)	MLP	ANT [30]	Random Forests	GBDTs
sarcos	**1.23±0.05**	2.36±0.16	1.38*	2.37±0.01	1.44±0.01

Table 2: Model size (MB), total inference / training time (s) and accuracy (%) comparison. Number of trees is set to 100 for random forests, GBDTs, ThunderGBM random forests (RFs) and GBDTs for faster speed and smaller size.

	Method	Model Size	Time		Acc
			Inference	Training	
gisette	NFL (ours)	35	0.17	62.51	**97.82**
	Random Forests	3.6	0.12	**0.67**	96.70
	ThunderGBM RFs	0.6	2.77	24.96	93.60
	GBDTs	**0.2**	**0.01**	181.14	96.70
	ThunderGBM GBDTs	0.6	2.04	18.78	91.79
mnist	NFL (ours)	9.6	**0.17**	194.45	**98.42**
	Random Forests	137	0.31	**2.09**	96.85
	ThunderGBM RFs	6.1	0.76	19.43	93.16
	GBDTs	**1.7**	0.42	2877.78	94.87
	ThunderGBM GBDTs	6.1	0.99	23.66	93.78
letter	NFL (ours)	**3.4**	**0.19**	43.49	**97.78**
	Random Forests	106	0.39	**0.38**	96.12
	ThunderGBM RFs	15.9	0.35	16.03	93.29
	GBDTs	4.5	0.27	50.88	92.04
	ThunderGBM GBDTs	15.9	0.33	15.20	92.99

accuracy on gisette but the model size is larger than other forest methods, indicating that NFL may be unfriendly to those datasets with non-sparse high dimensions in terms of model size. Note that although we use smaller $nMul$ values in Table 2 than the experiments reported in Table 1, NFL's accuracy in Table 2 are still similar to those in Table 1. Effect of NFL's hyper-parameters such as $nMul$ will be studied using ablative experiments.

4.3. Ablation studies

We choose the 4 *largest* datasets among those 6, i.e., gisette, mnist, letter and usps, for ablative experiments. Ablation studies include three parts: expansion rate, number of channels per group and expansion height/width.

Expansion rate. As is known in random forests or GBDTs, we can increase the number of decision trees to boost performance. Similarly, we can increase $nMul$ in NFL to make our forest larger and we conduct ablation studies about $nMul$. Here we set $nPer$ to 1 for all experiments and other settings remain the same as in the previous subsection. The results in Figure 3a show that when $nMul$ grows, the average accuracy increases and the standard deviation becomes smaller. It indicates that as $nMul$ increases, more trees are integrated into our model and the performance becomes better and our model gets more robust.

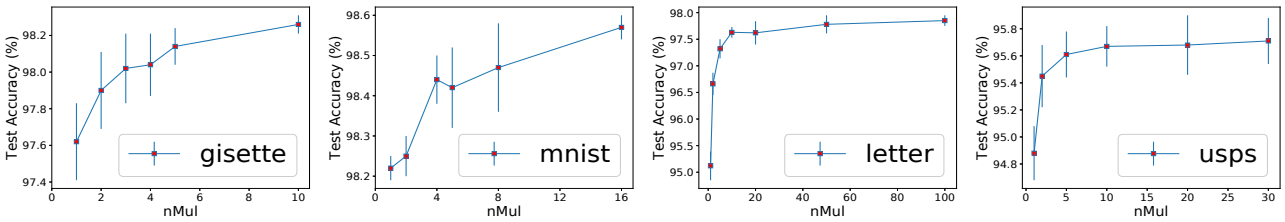
Number of channels per group. We can also increase $nPer$ to increase the number of features utilized in each tree. Here we set $nMul$ to 1 for all experiments and other settings remain the same. The results in Figure 3b show that when $nPer$ grows, the test accuracy will increase at first and then it will become stable or slightly decrease. It means that as $nPer$ increases, the capacity of each tree and hence the whole model will also increase, thus the accuracy will also increase at first. However, the model is more likely to overfit with large $nPer$ and model capacity and we can see that the performance will not continue to improve.

Expansion height/width. We set $nMul$ to 1, 5 and 50 for gisette, mnist and letter and we set $nPer$ to 1 for all these datasets. The results in Figure 3c show that when dH/dW is very small, i.e., equals 1, the result is bad, especially for gisette. When dH/dW grows, the result becomes better and will not continue to improve when it grows beyond 3. Therefore, 2 or 3 is a good choice for dH/dW in terms of accuracy and efficiency.

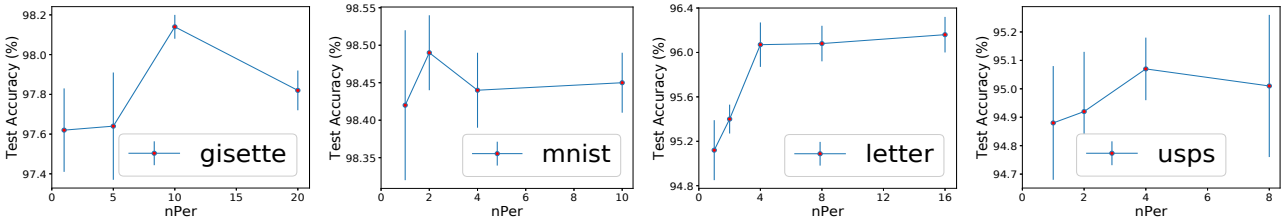
4.4. Fine-grained Visual Categorization

We then evaluate NFL in CNN architectures for image recognition. NFL is used after GAP to non-linearly transform the GAP output vector at the end of the network. First, this section evaluates NFL with ResNet-50 [9] and VGG-16 [28] on the Birds, Aircrafts and Cars datasets. We compare our method with baseline models and one representative higher-order pooling method.

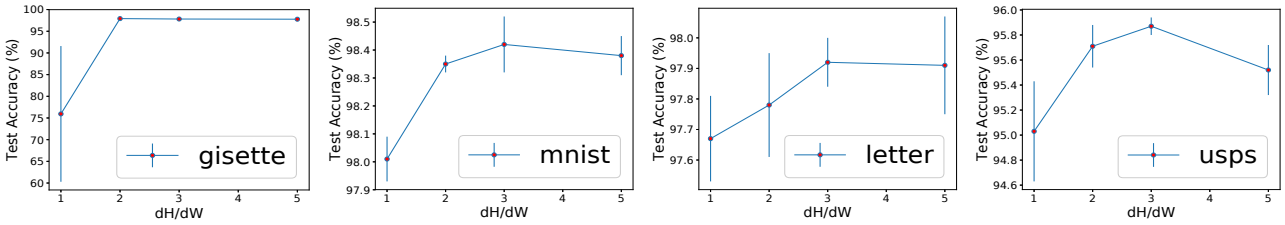
Implementation details: For fair comparisons, we fol-



(a) Classification results using different expansion rates $nMul$.



(b) Classification results using different number of channels per group $nPer$.



(c) Classification results using different expansion size dH/dW.

Figure 3: Ablation studies of $nMul$, $nPer$ and dH/dW on gisette, mnist and letter (from left to right in each sub figure). We plot the average accuracy and standard deviation of 5 trails at each point.

low [21] for experimental setting and evaluation protocal. We crop 448×448 patches as input images for all datasets. For baseline models, we replace the 1000-way softmax layer of ResNet-50 pretrained on ImageNet ILSVRC-12 [26] with a k-way softmax layer for finetuing, where k is the number of classes in the fine-grained dataset. We replace all fc layers of pretrained VGG-16 with a GAP layer plus a k-way softmax layer to fit 448×448 input. We finetune all the networks using SGD with batch size of 32, a momentum of 0.9 and a weight decay of 0.0001. We train the networks for 65 epochs, initializing the learning rate to 0.002 which is devided by 10 every 20 epochs. For NFL models, we replace the 1000-way softmax layer of pretrained ResNet-50 with our NFL module, specifically, random permutations, 1 group convolution layer and a k-way softmax layer (k is number of classes), which is called ResNet-50+NFL. Here we set $nMul$, $nPer$ and dH/dW to 2, 64 and 3, respectively. Moreover, we also use the pretrained VGG-16 as our backbone network to construct VGG-16+NFL in a similar way, except that we set $nMul$ to 6 considering different feature dimensions. We finetune our models under the same setting as the baseline models. These models integrating NFL are trained end-to-end as the baseline models.

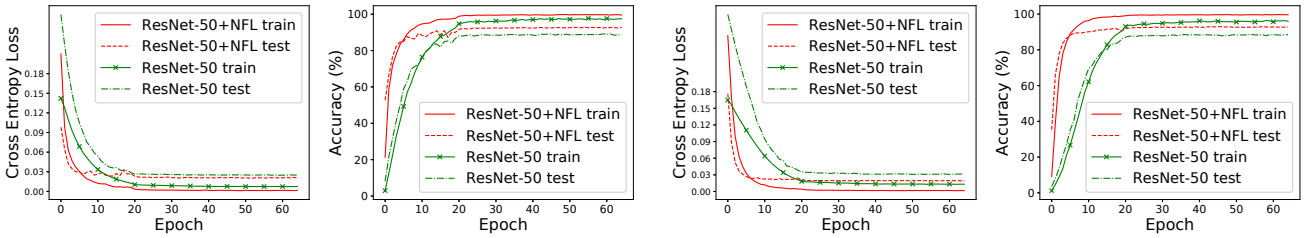
Comparison among different algorithms: Table 3 shows that our NFL method achieves significant improvement compared to baseline models, with negligible increase in parameters, FLOPs and real running time. It is worth mentioning that VGG-16+NFL achieves $7.2\%, 8.3\%$ and 9.3% relative improvement over baseline models on Birds, Aircrafts and Cars, respectively. Besides, our NFL performs consistently better than B-CNN [21] on all the 3 datasets under the VGG-16 architecture despite using much fewer parameters, FLOPs and real running time. Furthermore, the learning curves in Figure 4 shows that NFL can greatly accelerate the convergence and achieves better results both in accuracy and convergence speed than baseline methods (the red curves v.s. the green curves). It indicates that NFL can effectively learn non-linear feature representations and achieves good results on fine-grained recognition.

4.5. ImageNet ILSVRC-12

We then evaluate NFL on the large-scale ImageNet ILSVRC-12 task and also NFL is used after GAP at the end

Table 3: Comparison of representation dimensions, parameters, FLOPs, inference time per image (ms) and accuracy (%) on fine-grained benchmarks. The inference time is recorded with batch size of 1 on both CPU and GPU.

Method		# Dim	# Params	# FLOPs	Inference Time		Accuracy		
					CPU	GPU	Birds	Aircrafts	Cars
ResNet-50	Baseline	2K	23.92M	16.53G	540.48	28.16	84.0	88.6	89.2
	ResNet-50+NFL (ours)	4K	26.70M	16.53G	541.57	28.38	**86.7**	**92.8**	**93.4**
VGG-16	Baseline	0.5K	15.34M	61.44G	644.18	28.24	78.7	82.7	83.7
	B-CNN ([21])	262K	67.14M	61.75G	856.46	31.90	84.0	84.1	90.6
	VGG-16+NFL (ours)	3K	17.11M	61.44G	645.19	28.58	**84.4**	**89.6**	**91.5**



(a) Learning curves of Aircrafts

(b) Learning curves of Cars

Figure 4: Loss and accuracy learning curves. Both ResNet-50 and ResNet-50+NFL are trained under the same setting.

of the network.

Implementation details: We train a ResNet-50+NFL model from scratch on ImageNet, which is described in the previous subsection except that the last layer is a 1000-way softmax layer. The images are resized with shorter side=256, then a 224×224 crop is randomly sampled from the resized imgae with horizontal flip and mean-std normalization. Then, the preprocessed images are fed into ResNet-50+NFL model. We train ResNet-50+NFL using SGD with batch size of 256, a momentum of 0.9 and a weight decay of 1e-4 for 100 epochs. The initial learning rate starts from 0.1, and is devided by 10 every 30 epochs. A ResNet-18+NFL model is constructed and trained in a similar way, except that we set $nMul$ and $nPer$ to 4 and 32, respectively. For MobileNetV2 [27], we set $nMul$ and $nPer$ to 1 and 32, respectively. We train the network using SGD with batch size of 256, a momentum of 0.9 and a weight decay of 4e-5 for 150 epochs. We initialize the learning rate to 0.05 and use cosine learning rate decay.

Comparison with baseline methods: Table 4 shows that NFL produces 0.70%, 1.92% and 0.73% top-1 error (1-crop) less than the original MobileNetV2, ResNet-18 and ResNet-50 model, respectively, with negligible increase in parameters and FLOPs. It is worth noting that if we use larger dH/dW, $nMul$ and $nPer$, we can get better results at the cost of more parameters and FLOPs, which are included in the appendix. It indicates that our NFL method is also effective for large scale recognition, achieving better performance consistently under various architectures.

Table 4: Error rate (%, 1-crop prediction) comparison on ImageNet ILSVRC-12 under different architectures.

Method	# Params	# FLOPs	Top-1 / 5 err.
Original ResNet-50[1]	25.56M	4.14G	23.85 / 7.13
ResNet-50+NFL (ours)	29.98M	4.14G	**23.12** / **6.62**
Original ResNet-18[1]	11.69M	1.82G	30.24 / 10.92
ResNet-18+NFL (ours)	13.82M	1.83G	**28.32** / **9.77**
Original MobileNetV2[1]	3.50M	0.33G	28.12 / 9.71
MobileNetV2+NFL (ours)	3.88M	0.33G	**27.42** / **9.39**

[1] https://pytorch.org/docs/master/torchvision/models.html

4.6. NFL across all layers

Motivated by the Squeeze-and-Excitation (SE) method [12], we use NFL to replace all the SE modules. We conduct experiments on CIFAR-10 [17], CIFAR-100 [17] and the ImageNet ILSVRC-12 task. We compare our method with baseline methods and SENet under various architectures.

Implementation details: We replace the 2 fc layers in each SE block with NFL, specifically, random permutations and 1 group convolution layer followed by sigmoid activation, as shown in Figure 5, which is called SENet+NFL. In all our experiments in this section, we set $nMul$ and $nPer$ to 1 and dH/dW to 3 for SENet+NFL and the reduction ratio is set to 16 for SENet as is done in [12]. For CIFAR-10 and CIFAR-100, we use ResNet-20 [9], ResNet-50 [9] and Inception-v3 [29] as the backbone network. Mean subtraction, horizontal random flip and 32×32 random crops after

Table 5: Comparison of params, FLOPs and accuracy (%) on CIFAR-10 and CIFAR-100 under various architectures.

Method		CIFAR-10			CIFAR-100		
		# Params	#FLOPs	Accuracy	# Params	#FLOPs	Accuracy
ResNet-20	original ResNet-20	0.27M	41.62M	92.75	0.28M	41.63M	69.33
	SE-ResNet-20	0.27M	41.71M	93.28	0.28M	41.72M	70.35
	SE-ResNet-20+NFL (ours)	0.28M	41.71M	**93.73**	0.28M	41.72M	**70.38**
ResNet-50	original ResNet-50	23.52M	1311.59M	95.78	23.71M	1311.96M	80.41
	SE-ResNet-50	26.04M	1318.42M	95.59	26.22M	1318.79M	**81.57**
	SE-ResNet-50+NFL (ours)	23.67M	1313.56M	**96.05**	23.86M	1313.93M	81.48
Inception-v3	original Inception-v3	22.13M	3411.04M	94.83	22.32M	3411.41M	79.62
	SE-Inception-v3	23.79M	3416.04M	95.60	23.97M	3416.41M	80.44
	SE-Inception-v3+NFL (ours)	22.23M	3412.85M	**95.67**	22.42M	3413.22M	**80.54**

Table 6: Comparison of parameters, FLOPs, inference time per image (ms) and error rate (%, 1-crop prediction) comparison on ImageNet ILSVRC-12 under SENet architectures. The inference time is recorded with batch size of 1 on both CPU and GPU.

Method	# Params	# FLOPs	Inference Time		Top-1 / 5 err.	
			CPU	GPU	reported in [12]	our re-implementation
Original ResNet-50[1]	25.56M	4.14G	465.39	21.07	24.80 / 7.48	23.85 / 7.13
SE-ResNet-50	28.07M	4.15G	581.32	35.82	23.29 / 6.62	**22.68 / 6.30**
SE-ResNet50+NFL (ours)	25.71M	4.14G	523.97	32.80	- / -	22.89 / 6.57

[1] https://pytorch.org/docs/master/torchvision/models.html

padding 4 pixels on each side were performed as data preprocessing and augmentation. We train all networks from scratch using SGD with 0.9 momentum, a weight decay of 5e-4 and batch size of 128 for 350 epochs. The initial learning rate starts from 0.1 using cosine learning rate decay. For ImageNet, we follow the same setting as in [12]. The images are resized with shorter side=256, then a 224×224 crop is randomly sampled from the resized imgae with horizontal flip and mean-std normalization. We use SGD with a momentum of 0.9, a weight decay of 1e-4, and batch size of 256 and the initial learning rate is set to 0.15 and decreased by a factor of 10 every 30 epochs. Models are trained for 100 epochs from scratch.

Comparison with baseline models: Table 5 shows that under ResNet-20, SENet+NFL achieves the highest accuracy on CIFAR-10 and CIFAR-100 with negligible increase in parameters and FLOPs. For ResNet-50 and Inception-v3 backbone, SENet+NFL achieves comparable or better accuracy than original SENet despite using fewer parameters and FLOPs, further demonstrating the effectiveness of NFL.

Table 6 shows that under ResNet-50, SENet+NFL achieves fewer parameters, FLOPs and real running time than original SENet while maintaining comparable accuracy. SENet+NFL also achieves higher accuracy than the baseline method with negligible increase in parameters and FLOPs. It indicates that NFL can be integrated not only at the end of a CNN as shown in the previous sections but also across all layers in a CNN to learn non-linear mapping effectively.

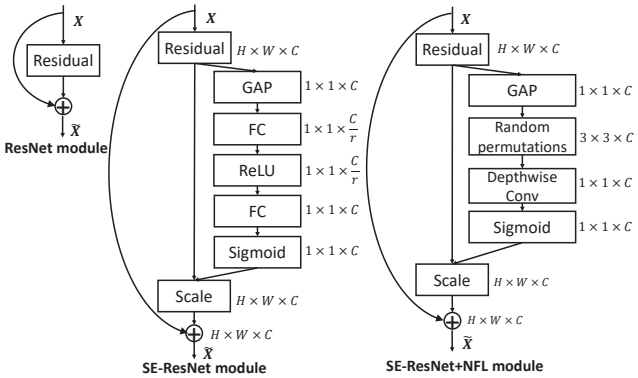


Figure 5: The schema of the original Residual module (left), the SE-ResNet module (middle) and the SE-ResNet+NFL module (right).

5. Conclusions

We proposed a deep learning based random-forest-like method NFL. We introduced the feature bagging process into deep learning with random permutations acting as resampling and group convolutions acting as aggregation, where each base learner learns from a subset of features. NFL can handle vectorized inputs well and can be installed into CNNs seamlessly both at the end of the network and

across all layers in the network. On one hand, it enriches forest method with the capability of end-to-end representation learning as well as pervasive deep learning software and hardware support. On the other hand, it effectively learns non-linear feature representations in CNNs with negligible increase in parameters, FLOPs and real running time. We have successfully confirmed the effectiveness of NFL on standard machine learning datasets, popular CIFAR datasets, challenging fine-grained benchmarks as well as the large-scale ImageNet dataset. In the future, we will continue exploration on combining deep learning and traditional forest learning algorithms to better understand the relation between different approaches. Furthermore, we will extend NFL to handle datasets with high dimensional sparse features, as well as small datasets.

References

[1] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. 1, 2

[2] Mircea Cimpoi, Subhransu Maji, and Andrea Vedaldi. Deep filter banks for texture recognition and description. In *CVPR*, pages 3828–3836, 2015. 2

[3] André Elisseeff and Jason Weston. A kernel method for multi-labelled classification. In *NIPS*, pages 681–687, 2002. 4

[4] Ji Feng, Yang Yu, and Zhi-Hua Zhou. Multi-layered gradient boosting decision trees. In *NIPS*, pages 3551–3561, 2018. 2

[5] Jerome H. Friedman. Greedy function approximation: a gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001. 1, 2

[6] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, pages 580–587, 2014. 1

[7] Isabelle Guyon, Steve Gunn, Asa Ben-Hur, and Gideon Dror. Result analysis of the NIPS 2003 feature selection challenge. In *NIPS*, pages 545–552, 2005. 4

[8] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *ICCV*, pages 2961–2969, 2017. 1

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 1, 4, 5, 7

[10] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, and Joaquin Quiñonero Candela. Practical lessons from predicting clicks on ads at facebook. In *ADKDD*, pages 1–9, 2014. 2

[11] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002. 4

[12] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-excitation networks. In *CVPR*, pages 7132–7141, 2018. 2, 4, 7, 8

[13] Jonathan J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554, 1994. 4

[14] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *CVPR*, pages 3304–3311, 2010. 2

[15] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Buló. Deep neural decision forests. In *ICCV*, pages 1467–1475, 2015. 2, 4, 5

[16] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *ICCV Workshops*, pages 554–561, 2013. 2, 4

[17] Alex Krizhevsky and Geoffrey E. Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 4, 7

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012. 1

[19] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 4

[20] Peihua Li, Jiangtao Xie, Qilong Wang, and Zilin Gao. Towards faster training of global covariance pooling networks by iterative matrix square root normalization. In *CVPR*, pages 947–955, 2018. 2

[21] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear CNN models for fine-grained visual recognition. In *ICCV*, pages 1449–1457, 2015. 1, 2, 6, 7

[22] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, pages 3431–3440, 2015. 1

[23] David G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, pages 1150–1157, 1999. 2

[24] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. Technical report, 2013. 2, 4

[25] Florent Perronnin, Jorge Sanchez, and Thomas Mensink. Improving the Fisher kernel for large-scale image classification. In *ECCV*, volume 6314 of *LNCS*, pages 143–156. Springer, 2010. 2

[26] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015. 2, 4, 6

[27] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *CVPR*, pages 4510–4520, 2018. 4, 7

[28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, pages 1–14, 2015. 1, 4, 5

[29] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, pages 2818–2826, 2016. 4, 7

[30] Ryutaro Tanno, Kai Arulkumaran, Daniel C. Alexander, Antonio Criminisi, and Aditya Nori. Adaptive neural trees. In *ICML*, pages 6166–6175, 2019. 4, 5

[31] Amber Thomas. Kaggle 2017 survey results. Website, 2017. https://www.kaggle.com/amberthomas/kaggle-2017-survey-results. 1

[32] Sethu Vijayakumar and Stefan Schaal. An o(n) algorithm for incremental real time learning in high dimensional space. In *ICML*, pages 288–293, 2000. 4

[33] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011. 2, 4

[34] Wen Zeyi, Shi Jiashuai, He Bingsheng, Li Qinbin, and Chen Jian. ThunderGBM: Fast GBDTs and random forests on GPUs. Website, 2019. https://github.com/Xtra-Computing/thundergbm. 2

[35] Zhi-Hua Zhou and Ji Feng. Deep forest: Towards an alternative to deep neural networks. In *IJCAI*, pages 3553–3559, 2017. 2

Table 7: Attributes of the machine learning datasets. The above 6 datasets are classfication datasets and sarcos is a regression dataset.

Datasets	# Category	# Training	# Testing	# Dim
satimage	6	4435	2000	36
gisette	2	6000	1000	5000
mnist	10	60000	10000	780
letter	26	15000	5000	16
usps	10	7291	2007	256
yeast	14	1500	917	8
sarcos	-	44484	4449	21

A. Machine learning datasets

A brief description of the machine learning datasets including the train-test split, the number of categories and feature dimensions is given in Table 7. Note that sarcos is a regression dataset so it doesn't have the attribute of the category.

B. Speed comparison experimental details

For better trade-off between model size, speed and accuracy, we reduce the number of trees for random Forests and GBDTs and the parameter $nMul$ for NFL correspondingly. In Table 2, we use the same setting as in Table 1 except that we set $nMul$ to 1, 5 and 50 for NFL for gisette, mnist and letter, respectively, for better model size, speed and accuracy trade-off. Note that although we use smaller $nMul$ values in Table 2 than the experiments reported in Tabel 1, NFL's accuracies in Tabel 2 are still similar to those in Table 1 (e.g., 97.85 in Table 1 v.s. 97.78 in Table 2 on letter). Correspondingly, we reduce the number of trees to 100 for random forests and GBDTs for faster speed and smaller size. We also use 5-fold cross validation on the train set and choose the best value for other parameters. Note that although we reduce the number of trees for random forests from 500 to 100 on mnist, the accuracy drops slightly (from 96.96% to 96.85%) while the model size is reduced by 5 times (from 680M to 137M). We record total training time on the train set and inference time on the test set in seconds.

C. More results

If we use larger dH/dW values, we can use more group convolution layers to make our forest deeper. We set dH/dW to 3 and use one group convolution layer of kernel size (3,3) in the paper. Here we set dH/dW to 5 and use two group convolution layers of kernel size (3,3) and we compare the results under these two settings.

Table 8: Error rate (%, 1-crop prediction) comparison on ImageNet ILSVRC-12 under different architectures.

Method	# Params	# FLOPs	Top-1 / 5 err.
Original ResNet-50[1]	25.56M	4.14G	23.85 / 7.13
ResNet-50+NFL (2 layers)	37.07M	4.19G	**23.08 / 6.22**
Original ResNet-18[1]	11.69M	1.82G	30.24 / 10.92
ResNet-18+NFL (2 layers)	20.02M	1.86G	**29.03 / 10.03**
Original MobileNetV2[1]	3.50M	0.33G	28.12 / 9.71
MobileNetV2+NFL (2 layers)	7.75M	0.35G	**27.02 / 8.94**

[1] https://pytorch.org/docs/master/torchvision/models.html

Table 9: Error rate(%, 10-crop prediction) comparison of ResNet-50+NFL with the original PyTorch ResNets on ImageNet-ILSVRC12.

Method	# Params	# FLOPs	Top-1 / 5 err.
Original ResNet-50[1]	25.56M	4.14G	22.57 / 6.24
ResNet-50+NFL (2 layers)	37.07M	4.19G	**21.44 / 5.72**
Original ResNet-101 [1]	44.55M	7.87G	21.06 / 5.55

[1] https://pytorch.org/docs/master/torchvision/models.html

C.1. Fine-grained Visual Categorization

First, we conduct experiments on fine-grained benchmarks for NFL with 2 group convolution layers.

Implementation details: When $dH/dW = 5$, our NFL module consists of random permutations, 2 group convolution layers and a k-way softmax layer (k is number of classes). And we remain all other settings the same as described in Sec 4.4.

Comparison among different settings: From Table 10 we can see that when we use more group convolution layers, we can get better results on all the 3 fine-grained benchmarks under VGG-16 at the cost of more parameters, FLOPs and real running time.

C.2. ImageNet ILSVRC-12

Then, we evaluate NFL with 2 group convolution layers on the large-scale ImageNet ILSVRC-12 task.

Implementation details: Due to resource constraints, we finetune a ResNet-50+NFL model on ImageNet, which is described in the previous subsection except that the last layer is a 1000-way softmax layer and we set $nPer$ to 128. We finetune ResNet-50+NFL using SGD with batch size of 256, a momentum of 0.9 and a weight decay of 0.0005 for 30 epochs. The initial learning rate starts from 0.001, and is devided by 10 every 10 epochs. A ResNet-18+NFL model is constructed and finetuned in a similar way, except that we set $nMul$ and $nPer$ to 8 and 64, respectively. For MobileNetV2, we set $nMul$ and $nPer$ to 2 and 64, respectively. We finetune the network for 45 epochs with using cosine learning rate decay.

Comparison with baseline methods: Table 8 shows

Table 10: Comparison of representation dimensions, FLOPs, inference time per image (ms) and accuracy (%) on fine-grained benchmarks. The inference time is recorded with batch size of 1 on both CPU and GPU.

Method		# Dim	# Params	# FLOPs	Inference Time		Accuracy		
					CPU	GPU	Birds	Aircrafts	Cars
ResNet-50	Baseline	2K	23.92M	16.53G	540.48	28.16	84.0	88.6	89.2
	NFL (1 group conv layer)	4K	26.70M	16.53G	541.57	28.38	**86.7**	**92.8**	**93.4**
	NFL (2 group conv layers)	4K	29.07M	16.55G	543.04	30.77	86.6	**92.8**	93.1
VGG-16	Baseline	0.5K	15.34M	61.44G	644.18	28.24	78.7	82.7	83.7
	B-CNN	262K	67.14M	61.75G	856.46	31.90	84.0	84.1	90.6
	NFL (1 group conv layer)	3K	17.11M	61.44G	645.19	28.58	84.4	89.6	91.5
	NFL (2 group conv layers)	3K	18.89M	61.46G	645.72	30.13	**84.8**	**89.7**	**91.6**

that NFL produces 1.10%, 1.21% and 0.77% top-1 error (1-crop) less than the original MobileNetV2, ResNet-18 and ResNet-50 model, respectively. Table 9 shows that, compared to the original PyTorch ResNets, with 10-crop prediction our method performs 1.13% better than ResNet-50, while being comparable to ResNet-101. It indicates that our NFL method is also effective for large scale recognition, achieving performance matching deeper CNNs with much shallower one. These results indicate that we can get better results when using more layers as well as setting $nMul$ and $nPer$ larger at the cost of more parameters and FLOPs.