2018 24th International Conference on Pattern Recognition (ICPR)
Beijing, China, August 20-24, 2018

Retraining: A Simple Way to Improve the Ensemble Accuracy of Deep Neural Networks for Image Classification

Kaikai Zhao
Graduate School of Systems Life Sciences
Kyushu University, Japan
Email: cyoukaikai@gmail.com

Tetsu Matsukawa, Einoshin Suzuki
Department of Informatics, ISEE
Kyushu University, Japan
Email: {matsukawa, suzuki}@inf.kyushu-u.ac.jp

***Abstract*—In this paper, we propose a new heuristic training procedure to help a deep neural network (DNN) repeatedly escape from a local minimum and move to a better local minimum. Our method repeats the following processes multiple times: randomly reinitializing the weights of the last layer of a converged DNN while preserving the weights of the remaining layers, and then conducting a new round of training. The motivation is to make the training in the new round learn better parameters based on the "good" initial parameters learned in the previous round. With multiple randomly initialized DNNs trained based on our training procedure, we can obtain an ensemble of DNNs that are more accurate and diverse compared with the normal training procedure. We call this framework "retraining". Experiments on eight DNN models show that our method generally outperforms the state-of-the-art ensemble learning methods. We also provide two variants of the retraining framework to tackle the tasks of ensemble learning in which 1) DNNs exhibit very high training accuracies (e.g., $> 95\%$) and 2) DNNs are too computationally expensive to train.**

I. Introduction

One central problem in classification tasks is how to train a classifier that will perform well not only on training data but also on unseen data. The performance of the trained classifier is typically measured by the classification accuracy. Deep neural networks (DNNs) have exhibited the highest classification accuracies for various image classification tasks in recent years [1].

The training of DNNs refers to optimizing a loss function with respect to the parameter set Θ, which typically consists of a huge number of parameters. For instance, the AlexNet [2], which is an eight-layered network, consists of 60+ millions of parameters. Stochastic gradient descent (SGD) is the dominant method for training a DNN [3], as it explores the parameter space very efficiently. However, training a DNN using SGD with a single round typically converges to a local minimum, even if it often exhibits a high classification accuracy [4]. How to make the DNN converge to a better local minimum is still a very challenging task [5], [6].

One way of escaping from a local minimum is to use a large learning rate then decrease it to find a new local minimum. Another way is to randomly perturb Θ (e.g., $\Theta + \delta\Theta$) of the converged DNN so that Θ moves to a neighboring region in the parameter space, and then restart the training. However, since these methods explore the parameter space somehow "blindly", the obtained new local minima are usually as good as (not better than) the previous ones.

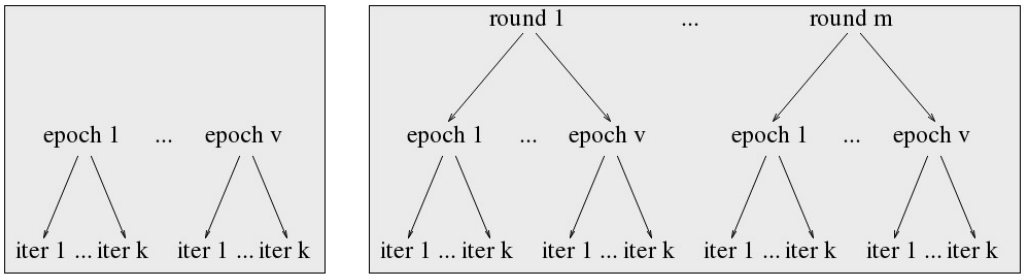
A heuristic method is expected to provide a "good direction" when exploring a new local minimum. It is well established that DNN generally learns generic features (e.g., edge detectors or color blob detectors) in its lower layers and task-specific features (e.g., parts of specific objects) in its upper layers [7]. The features in the upper layers are (weighted) combinations of the features in the lower layers and they are directly responsible for making predictions. Given a converged DNN and its learned features, if we can learn better (in terms of the representative ability) upper-layer features, then it is very likely that we can find a slightly better local minimum. Here the better upper-layer features serve as a sense of a "good direction".

Similar ideas that pay special attention to the upper-layer parameters have proven to be effective. For instance, it has been reported that small but consistent improvements of the classification accuracy were achieved by simply replacing the softmax layer (the last layer of a DNN) with a linear support vector machine (SVM) [9] when training a DNN [8]. The improvements are mainly due to the fact that SVMs generally learn better boundaries than softmax.

In this paper, we propose a new heuristic training method which exploits the idea of learning better upper-layer parameters for a converged DNN. Our method repeats the following processes multiple times: randomly reinitializing the weights of the last layer of a converged DNN while preserving the weights of the remaining layers, and then conducting a new round of training. The motivation is to make the training in the new round learn better parameters based on the "good" initial parameters learned in the previous round.

The structures of the standard method and our method for training a DNN are illustrated in Fig. 1. The standard method is of two levels: a single *round* of training with multiple *epochs*, and each epoch consists of multiple *iterations*. Usually, we use a mini-batch of training examples to estimate the gradients. The number of passes of the mini-batches corresponds to the number of iterations, and a single pass of the entire training examples corresponds to an epoch. A round of training means a number of passes of the entire training examples until the training loss converges. Our method is of three levels: multiple

978-1-5386-3788-3/18/$31.00 ©2018 IEEE



(a) standard method (b) our method

Fig. 1. Structures of the standard method and our method.

rounds of training, in which each round consists of multiple epochs and each epoch consists of multiple iterations.

The main contributions of this paper are as follows. 1) We show that training a DNN in multiple rounds with the weights of its last layer being reinitialized at the beginning of each round generally results in better local minima. 2) We propose a new ensemble learning method which exploits the better local minima.

II. Related Work

The most relevant work to our method is snapshot ensemble [10]. Snapshot ensemble repeatedly uses a large learning rate, decreases it to drive a converged DNN to different local minima and then forms an ensemble of the visited local minima. Each time a new local minimum is obtained, a round of training is ended. Note that though snapshot ensemble trains a DNN in multiple rounds, each of its base models is actually trained in a single round. Snapshot ensemble has two limitations. First, the later local minima can be worse than the earlier ones as snapshot ensemble explores the parameter space "blindly". Second, the diversity of the local minima is likely to be low as the local minima are all obtained by training a single DNN in such a way that the later one is obtained by directly training the previous one.

Greedy layer-wise pretraining [11], [12] trains a DNN in two rounds. In the first round, the weights of the DNN are initialized layer by layer. Early layers are trained first and then fixed when training later layers. In the second round, a joint training of all the layers is conducted. A similar method [13] trains several layers instead of one layer at a time to initialize the network in the first round. These methods are similar to ours in the sense that the previous round of training provides a good starting point for the later round of training. However, we train all the layers simultaneously rather than layer(s) by layer(s) to simplify the training processes. We conduct multiple rounds of training rather than only two rounds to keep improving the classification accuracy.

Fine-tuning [14] also trains a DNN in two rounds. It is often used to feed the generic features learned from one dataset D_1 to another dataset D_2. However, fine-tuning typically requires a large dataset D_1, which is not always available for various deep learning tasks. We use only one dataset.

III. Our Method: Retraining

To overcome the flaws of snapshot ensemble, we train N_M DNNs (instead of one) with different random initial weights to increase the diversity of the trained base models. We reinitialize the learned weights of the last layer rather than keeping all the learned weights to increase not only the diversity but also the classification accuracy. The increase of accuracy is likely to be achieved since the training in the new round concentrates on learning better weights for the last layer.

Retraining framework. The input to our method is a training dataset D_{train}, a DNN architecture and its training parameters (e.g., the initial learning rate and its decay policy, the maximum number of iterations, the weight decay and the momentum), which define how to train a single DNN. The outputs are the combined votes of a number of trained DNNs for a test dataset D_{test}.

Let n_l denote the number of layers in a network. The parameter set Θ consists of the weights of the network. That is, $\Theta = \{W_1, ..., W_{n_l-1}\}$, where W_l is the weight matrix that reflects the connectivity between the lth layer and $l+1$th layer. Note that there are $n_l - 1$ weight matrices for an n_l-layered network and the weights of the lth layer are W_{l-1} rather than W_l. SGD updates the weights in W_l of a DNN by $W_l = W_l - \alpha \nabla_{W_l}$, where ∇_{W_l} are the gradients of the loss function with respect to W_l and α is the global learning rate. Sometimes we may assign a layer-specific learning rate to W_l for various reasons (e.g., a better optimization [15] or a fast convergence [16]). Then the update of W_l becomes $W_l = W_l - \lambda_l \alpha \nabla_{W_l}$, where λ_l is the global learning rate multiplier for W_l.

The overall framework of retraining is shown in Fig. 2. It consists of two phases: training and testing. The training phase consists of a normal training procedure followed by multiple retraining procedures. Firstly, we randomly initialize N_M DNNs and conduct a normal training until convergence. Secondly, for each of the converged DNNs, we repeat the following steps 1) - 4) N_R rounds: 1) randomly reinitializing the weights W_{n_l-1} of the last layer, 2) setting the global learning rate multiplier λ_{n_l-1} of the new reinitialized layer to be b times as large as that in the normal training procedure (i.e., $\lambda_{n_l-1}^{retrain} = b\lambda_{n_l-1}^{normal}$), 3) reinitializing the global learning rate α to a relatively large value (i.e., $\alpha = \alpha_0$) and 4) training

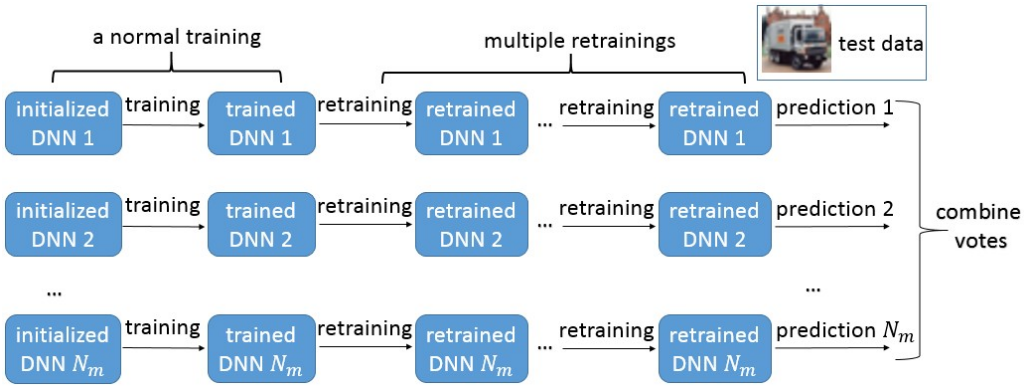


Fig. 2. The framework of retraining.

the DNN with decaying α.

Note that we only preserve the learned weights of the layers except the last one at the beginning of each round of retraining, rather than fixing them during the retraining processes. The latter option adds too strong constraints (i.e., most of the parameters are not allowed to be updated) to the optimization task, and thus a better local minimum is not likely to be obtained given such constraints. Our experiments show that fixing the weights increases neither the base model accuracy nor the base model diversity.

In the test phase, only the retrained DNNs at the end of the last round are used to make predictions for D_{test}. We use majority voting, due to its simplicity, to make the combined votes.

Justification of our method. There are two benefits of reinitializing the weights of the last layer rather than any other layer(s). First, reinitializing the last layer enables inheriting more well-learned features in the previous round, compared with reinitializing the early layers. Because the upper-layer features are combinations of the lower-layer features, reinitializing the lower-layer features will make all the subsequent later layers to rebuild the combinations. That is, if W_k is reinitialized, then $W_{k+1}, ..., W_{n_l-1}$ will have to make large adaptations to ensure that the feedforward propagation makes correct predictions. Especially, reinitializing the first layer means relearning the network from scratch. Second, reinitializing the last layer enables largely simplifying the optimization task in the new round, compared with reinitializing the last multiple layers. This fact is because most of the adaptations of the parameters in the new round of retraining are realized in the reinitialized layer(s). Thus with fewer layers reinitialized, the optimization task becomes simpler.

IV. Design of the Experiments

Datasets and DNN models. We use six datasets, MNIST [17], SVHN [18], CIFAR10 [19], CIFAR100 [19], FLOWER102 [20] and ILSVRC2012 [21]. We use the Caffe [22] framework. We use eight public off-the-shelf DNN models. Among them, MNIST-LeNet, CIFAR10-QUICK and CIFAR100-QUICK (slightly modified from CIFAR10-QUICK to fit the CIFAR100 dataset) are built-in models in the Caffe framework, while the other models (SVHN[1], CIFAR100-NIN[2], CIFAR100-DenseNet40[3], FLOWER102-AlexNet[4] and ILSVRC2012-NIN[5]) are public models available online. More details of the datasets and DNN models are shown in Table I.

Compared methods. We compare our retraining with: training only a single model (SM), the ensemble of DNNs trained with random initializations (RI) [23], Bagging [24], AdaBoost [25], error correcting output codes (ECOC) [26], N-ary ECOC [27], horizontal voting (HV) [28], snapshot ensemble (SE) [10], TreeNet [29] and DivLoss [30]. Usually, longer iterations produce better performances. Thus we also compare retraining with RI that uses the same number of iteration as retraining, which is called *RI-LongIters*.

Bagging [24] samples the training data with replacement and trains its base models using the sampled datasets. AdaBoost [25] trains a number of models sequentially and updates example weights based on correct/incorrect predictions in the earlier models so that the later models focus on the difficult examples in the training set. ECOC [26] forms an ensemble of binary classifiers by randomly merging the classes of a multi-class problem into two subsets (meta-classes, each of which is a super-class composed of classes). N-ary ECOC [27] is an extension of ECOC to the case of N meta-classes ($3 \leq N < N_C$, where N_C represents the number of classes). Horizontal voting [28] trains a single DNN until convergence and then continues the training with additional iterations, meanwhile it saves the snapshots of the DNN parameters along the path to obtain a pool of base models.

Settings of the compared methods. One important setting for an ensemble learning method is the number (N_M) of the base models to train, as some methods require a larger N_M than others to achieve their optimal performances. Among the compared methods, ECOC is the most demanding method for a large N_M, as each of its base models can only discriminate two meta-classes. It is reported in [31] that ECOC requires approximately $10 \log_2(N_C)$ base models to achieve its

[1]https://github.com/alexvking/neural-net-house-number-recognition
[2]https://gist.github.com/mavenlin/e56253735ef32c3c296d
[3]https://github.com/liuzhuang13/DenseNetCaffe
[4]http://jimgoo.com/flower-power/
[5]https://github.com/BVLC/caffe/wiki/Model-Zoo

TABLE I

STATISTICS OF THE TESTED DATASETS, AS WELL AS THE ARCHITECTURES AND THE TRAINING PARAMETERS OF THE TESTED DNN MODELS. HERE "*conv*" AND "*cccp*" REPRESENT CONVOLUTIONAL LAYERS, AND "*ip*" AND "*fc*" REPRESENT FULLY CONNECTED LAYERS. NOTE THAT N_C IS THE NUMBER OF CLASSES, *iter* IS THE ITERATION NUMBER AND max_iter IS THE MAXIMUM NUMBER OF ITERATIONS.

Dataset	Image Size	No. Train	No. Test	N_C	Model	Architecture	Weight decay	Batch size	max_iter	Global learning rate (α)
MNIST	28×28	60,000	10,000	10	MNIST-LeNet	conv1 → conv2 → ip1 → ip2	0.0005	64	10,000	$0.01 * (1 + 0.0001 * iter)^{-0.75}$
SVHN	32×32	73,257	26,032	10	SVHN	conv1 → conv2 → conv3 → ip1 → ip2	0.004	100	6,000	0.001 ($iter$: 1 ∼ 3,000), 0.0001 ($iter$: 3,001 ∼)
CIFAR10	32×32	50,000	10,000	10	CIFAR10-QUICK	conv1 → conv2 → conv3 → ip1 → ip2	0.004	350	5,000	0.001 ($iter$: 1 ∼ 4,000), 0.0001 ($iter$: 4,001 ∼)
CIFAR100	32×32	50,000	10,000	100	CIFAR100-NIN	conv1 → cccp1 → cccp2 → conv2 → cccp3 → cccp4 → conv3 → cccp5 → cccp6	0.0001	100	50,000	0.01 ($iter$: 1∼ 20,000), 0.001 ($iter$: 20,001 ∼)
					CIFAR100-QUICK	conv1 →conv2 → conv3 → ip1 → ip2	0.004	350	10,000	0.001 ($iter$: 1 ∼ 8,000), 0.0001 ($iter$: 8,001 ∼)
					CIFAR100-DenseNet40	conv1 → ... → conv39 → ip1	0.0001	64	230,000	0.1 ($iter$: 1 ∼ 115,000), 0.01 ($iter$: 115,001 ∼ 172,500), 0.001 ($iter$: 172,501 ∼)
FLOWER102	256×256	1,020	1,020	102	FLOWER102-AlexNet	conv1 → conv2 → conv3 → conv4 → conv5 → fc6 → fc7 → fc8	0.0005	50	12,000	0.001 ($iter$: 1 ∼ 10,000), 0.0001 ($iter$: 10,001 ∼)
ILSVRC2012	256×256	1,281,167	50,000	1,000	ILSVRC2012-NIN	conv1 → cccp1 → cccp2 → conv2 → cccp3 → cccp4 → conv3 → cccp5 → cccp6 → conv4 → cccp7 → cccp8	0.0005	128	450,000	0.01 ($iter$: 1 ∼ 200,000), 0.001 ($iter$: 200,001 ∼ 400,000), 0.0001 ($iter$: 400,001 ∼)

TABLE II

SETTINGS FOR N-ARY ECOC, HORIZONTAL VOTING (HV), SNAPSHOT ENSEMBLE (SE) AND RETRAINING.

Model	N-ary ECOC (tested N)	HV (snapshot interval)	SE and Retraining (α and max_iter)
MNIST	3, 4, 5, 7	1,000	same as in Table I
SVHN	3, 4, 5, 7	1,000	same as in Table I
CIFAR10-QUICK	3, 4, 5, 7	500	same as in Table I
CIFAR100-NIN	3, 5, 25, 33, 50, 75, 95	1,000	$max\_iter \leftarrow 30,000$; α: 0.01 ($iter$: 1 ∼ 20,000), 0.001 ($iter$: 20,001 ∼)
CIFAR100-QUICK	3, 4, 5, 10, 25, 33, 50, 75, 95	500	same as in Table I
CIFAR100-DenseNet40	-	1,000	$max\_iter \leftarrow 30,000$; α: 0.1 ($iter$: 1 ∼ 15,000), 0.01 ($iter$: 15,001 ∼ 22,500), 0.001 ($iter$: 22,501 ∼)
FLOWER102-AlexNet	3, 5, 10, 34, 50, 75, 95	500	same as in Table I
ILSVRC2012-NIN	-	5,000	$max\_iter \leftarrow 250,000$; α: 0.001 ($iter$: 1 ∼ 100,000), 0.0001 ($iter$: 100,001 ∼ 200,000), 0.00001 ($iter$: 200,001 ∼)

optimal performance for traditional classifiers such as SVM and decision tree. However, we observed that the required N_M for ECOC is slightly larger than $10\log_2(N_C)$ for DNNs. The larger N_M is probably because more stochastic processes (random initialization, SGD and local optimal solutions) are involved in training DNNs.

We set N_M large enough to ensure that every method achieves its optimal performance. We use $N_M = 100$ for both N_C = 10 ($10\log_2(N_C) \approx 33$) and $N_C \approx 100$ ($10\log_2(N_C) \approx 66$). However, the accuracy of MNIST-LeNet is very high (i.e., over 99%). There is little room for improvement in the classification accuracy by using ensembles as the base models will agree on over 99% of the predictions, and thus we use $N_M = 60$ instead. Because some DNNs are computationally expensive to train, we have to use a relatively small N_M (e.g., 60 for CIFAR100-NIN and FLOWER102-AlexNet), or even give up the tests of ECOC, N-ary ECOC, Bagging and AdaBoost for CIFAR100-DenseNet40 and ILSVRC2012-NIN.

In our method, we set the coefficient $b = 2.5$ for the learning rate of the reinitialized layer (based on empirical evidence). The experiments of evaluating different values for b, i.e., b = [0.1, 0.5, 1, 2.5, 5, 7.5, 10, 25, 50], are given in Section V-C.

For N-ary ECOC, we tested several typical values for N, for instance, N = [3, 4, 5, $N_C/20$, $N_C/10$, $N_C/5$, $N_C/3$, $N_C/2$, $9N_C/10$], and used the best one. For horizontal voting, we save the snapshots by every one or a few epochs after the training converges. The settings of the global learning rate α (initial value and decay policy) and the maximum number of iterations max_iter in each round of training are exactly the same for snapshot ensemble [10] and our method. For AdaBoost, we implemented AdaBoost.SAMMAR [32] because it is very effective in dealing with multi-class problems. More details of the settings for the compared methods are given in Table II.

Evaluation metrics. We use ensemble accuracy (EA) to evaluate different ensemble learning methods[6]. The EA, which is the accuracy of the combined votes, is often estimated based on either the predicted class labels or the posterior probabilities. We adopt the former due to its simplicity.

V. RESULTS AND ANALYSIS

A. Effective range of retraining

The results are shown in Table III. To help identify the effective range of retraining, we also report the average training accuracies (TAs) of the tested DNNs in the second column of Table III. We see that retraining outperforms all other methods

[6]The area under the ROC curve (AUC) is typically used in binary classification tasks with imbalanced class/classification costs. Our tested datasets differ from such problems and thus we do not evaluate the AUC.

TABLE III
AVERAGE TRAINING ACCURACIES (TAS) AND AVERAGE TEST ACCURACIES (SMS) OF A SINGLE MODEL, AND ENSEMBLE ACCURACIES OF DIFFERENT ENSEMBLE LEARNING METHODS.

Model	TA	SM	Ensemble Accuracy									
			RI	RI-LongIters	Our Method	Bagging	AdaBoost	ECOC	N-ary ECOC	HV	SE	Other
MNIST-LeNet$\times 60$	99.86	99.08	99.19	99.13	99.24 ($N_R=10$)	99.09	99.09	99.23	**99.27** ($N=3$)	99.13	99.19	-
SVHN$\times 100$	92.79	90.90	92.35	93.73	**94.83** ($N_R=10$)	92.51	91.65	93.12	93.22 ($N=3$)	91.95	92.48	-
CIFAR10-QUICK$\times 100$	87.24	77.19	82.38	83.70	**85.24** ($N_R=10$)	81.58	77.75	81.92	82.58 ($N=7$)	77.28	77.83	-
CIFAR100-NIN$\times 60$	91.00	59.84	67.42	68.06	**71.58** ($N_R=8$)	65.90	55.47	39.47	67.03 ($N=95$)	59.47	66.26	
CIFAR100-QUICK$\times 100$	64.80	44.81	57.03	59.47	**62.89** ($N_R=10$)	55.84	48.35	44.24	56.95 ($N=95$)	44.84	46.72	50.32 [30]
CIFAR100-DenseNet40$\times 10$	99.82	70.01	75.50	-	**75.85** ($N_R=4$)	-	-	-	-	70.38	73.55	-
FLOWER102-AlexNet$\times 60$	100.00	33.67	36.08	36.57	38.73 ($N_R=4$)	33.14	-	43.92	**45.59** ($N=3$)	33.04	34.61	-
ILSVRC2012-NIN$\times 10$	77.94	60.06	64.65	-	**66.03** ($N_R=4$)	-	-	-	-	60.62	61.53	65.69 [29]

in most cases. For instance, the EAs of retraining are 3.56% and 3.42% higher than those of the second best method on CIFAR100-NIN and CIFAR100-QUICK, respectively. Moreover, in general, the lower the TA, with a larger value retraining outperforms the second best method. Furthermore, for DNN models that retraining did not obtain the highest EAs, their TAs are extremely high (99.86% and 100.00% for MNIST-LeNet and FLOWER102-AlexNet, respectively).

The above experiments suggest that retraining is relatively more effective on DNNs which exhibit relatively low TAs (e.g., $< 95\%$) than DNNs which exhibit very high TAs (e.g., $> 95\%$). We provide the following hypothesis to explain the possible reason of the effective range. *Hypothesis 1: when the TA of a DNN is very high (e.g., $> 95\%$), which suggests that the DNN almost fully fits the training data, different DNNs tend to make similar predictions. Thus the ensemble learning methods that are relatively more effective in making diverse predictions achieve higher EAs.*

To verify *Hypothesis 1*, we estimate the diversities of the trained base models of retraining, ECOC and N-ary ECOC. Note that ECOC and N-ary ECOC outperformed retraining on the DNNs that exhibit very high TAs (as shown in Table III). The diversity is measured by the degree of difference in the predictions of base models on the test data. We use the Q statistic [33], which is one of the most frequently used diversity measures. Q statistic of the pairwise diversity between two classifiers C_i and C_j is given by

$$Q_{i,j} = \frac{N^{11}N^{00} - N^{01}N^{10}}{N^{11}N^{00} + N^{01}N^{10}}, \quad (1)$$

where N^{11} and N^{00} are the numbers of examples correctly and incorrectly predicted by both C_i and C_j, respectively; N^{01} and N^{10} are the numbers of examples correctly predicted by only C_i and only C_j, respectively. Q statistic, which is between [-1, 1], is smaller if the two classifiers are more diverse. The average Q statistic over all pairs of classifiers is given by

$$Q_{avg} = \frac{2}{N_M(N_M-1)} \sum_{i=1}^{N_M-1} \sum_{j=i+1}^{N_M} Q_{i,j}. \quad (2)$$

The results are shown in Table IV. We see that ECOC and N-ary ECOC always exhibit smaller Q_{avg} compared with retraining. The above observations are consistent with *Hypothesis 1*. The reason that ECOC or N-ary ECOC generally exhibits a high diversity is probably due to the fact that each of the meta-classes of the base models of ECOC or N-ary ECOC consists of a number of different classes. Take ECOC as an example, each meta-class consists of $N_C/2$ classes. Whatever the predicted meta-classes of two base models are for a test example, they often consist of a number of different classes, and thus the degree of difference in the predictions is large.

TABLE IV
Q STATISTIC (Q_{avg}) OF THE TRAINED BASE MODELS OF DIFFERENT ENSEMBLE LEARNING METHODS.

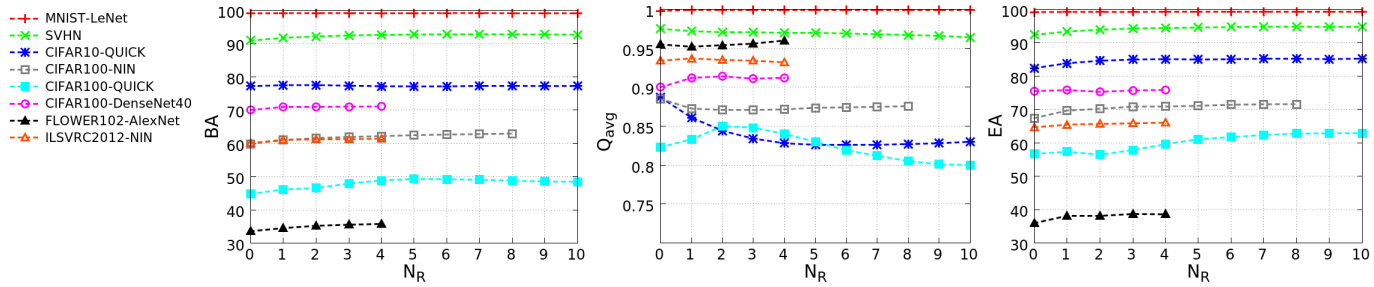
Model	Our Method	ECOC	N-ary ECOC
MNIST-LeNet$\times 60$	0.999	**0.955**	**0.974**
FLOWER102-AlexNet$\times 60$	0.960	**0.191**	**0.281**

B. Effects of multiple rounds of retraining

Because EA is closely related to base model accuracy (BA) and base model diversity, we plot the BA, Q_{avg} and EA with respect to the number (N_R) of retraining in Fig. 3 (a), (b) and (c), respectively, to investigate what happened during the multiple rounds of retraining.

For the BA, we see two interesting tendencies in Fig. 3 (a). First, the BAs generally increase as N_R increases. For instance, the BAs of ILSVRC2012-NIN are 60.06%, 61.06%, 61.18%, 61.28% and 61.37% for N_R = 0, 1, 2, 3 and 4, respectively. Second, the overall improvements of the BAs after multiple rounds of retraining over the BAs of the normal training can be large, even the average improvements of the BAs for each round may be small (e.g., 0.3%). For instance, the BAs after multiple rounds of retraining are 3.63%, 3.08%, 2.21%, 1.77%, 1.31% and 1.07% higher than the BAs of the normal training on CIFAR100-QUICK, CIFAR100-NIN, FLOWER102-AlexNet, SVHN, ILSVRC2012-NIN and CIFAR100-DenseNet40, respectively. These results suggest that the multiple rounds of retraining drive the converged DNNs to better local minima. The increases of BA may be due to the fact that the trained DNNs in the previous round provide a good starting point for the next round of retraining.

For the base model diversity, we see two different patterns in Fig. 3 (b). On the one hand, Q_{avg} generally decreases (i.e., diversity increases) as N_R increases for DNNs that exhibit relatively low TAs (e.g., $< 95\%$, SVHN, CIFAR10-QUICK, CIFAR100-NIN, CIFAR100-QUICK and ILSVRC2012-NIN). Basically, the reinitialization of W_{n_l-1} injects some randomness to the DNN base models and retraining perturbs their parameters multiple times, and thus the base models tend to make diverse predictions. Note that the decrease of Q_{avg} for



(a) BA vs. N_R (b) Q_{avg} vs. N_R (c) EA vs. N_R

Fig. 3. Average base model accuracies (BAs), Q statistic (Q_{avg}) and ensemble accuracies (EAs) of our method with respect to the number (N_R) of retraining. Note that N_R = 0 means that no retraining is conducted.

ILSVRC2012-NIN is small, which is probably because a large number of training examples might have a strong *averaging* effect on the training of DNNs since the update of the weights has to make compromises on all training examples.

On the other hand, Q_{avg} generally increases (i.e., diversity decreases) as N_R increases for DNNs that exhibit very high TAs (e.g., $> 95\%$, MNIST-LeNet, CIFAR100-DenseNet40 and FLOWER102-AlexNet). DNNs that exhibit very high TAs tend to make similar predictions (as we discussed in *Hypothesis 1*) and thus their diversities hardly increase. The improvements of BA (due to the multiple rounds of retraining) further increase the chance of making similar predictions for the base models as more predictions are correct and thus same.

Due to the above-observed patterns of the BA and the base model diversity, the EAs generally increase as N_R increases (as shown in Fig. 3 (c)). For instance, the EAs after multiple rounds of retraining are 5.86% and 4.16% higher than the EAs of the normal training on CIFAR100-QUICK and CIFAR100-NIN, respectively.

Regarding the question of how many rounds of retraining we should conduct for our method, we see in Fig. 3 (c) that, on the one hand, the EAs generally converge at N_R = 8 for DNNs that exhibit relatively low TAs (e.g., $< 95\%$), due to the increased BAs and diversities. On the other hand, the EAs generally converge at N_R = 3 for DNNs that exhibit very high TAs (e.g., $> 95\%$), due to the increased BAs but decreased diversities.

C. Effects of different values for the learning rate coefficient b

The value for b in our method affects the learning rate of the reinitialized layer (i.e., the last layer). For instance, b = 10 refers to setting the learning rate 10 times as large as that in the normal training procedure. Learning rate is a crucial parameter for training a DNN. A large learning rate might cause the parameters being updated too much so that the training diverges; A small learning rate might cause the learning making too slow progress or even getting stuck with a high cost value. Thus we need to appropriately set the value for b.

We conducted multiple rounds of retraining for the tested DNN models using b = 0.1, 0.5, 1, 2.5, 5, 7.5, 10, 25 and 50. The resulting BAs, Q_{avg} and EAs are shown in Fig. 4 (a), (b) and (c), respectively. For the BA, we see in Fig. 4 (a) that the highest BAs generally occur at around b = 1. Too small values for b (i.e., $b < 1$) or too large values for b (i.e., $b > 10$) tend to cause the BAs to be much lower (5%+) than the BAs of b = 1, e.g., b = 0.1 for CIFAR100-QUICK, b = 10, 25 and 50 for CIFAR100-DenseNet40 and b = 50 for MNIST. The above observations suggest that the BA is not stable when $b < 1$ or $b > 10$. It is probably because the former causes the reinitialized layer to learn too slow than the other layers, while the latter causes the reinitialized layer to learn too fast than the other layers.

For the Q_{avg}, we see in Fig. 4 (b) that, in general, the larger the value for b, the smaller the Q_{avg}, i.e., the more diverse the trained base models. One possible reason of the observed pattern is that a relatively large learning rate of the reinitialized layer is more likely to drive the parameters of the DNN to a region that is far from the previous local minimum in the parameter space.

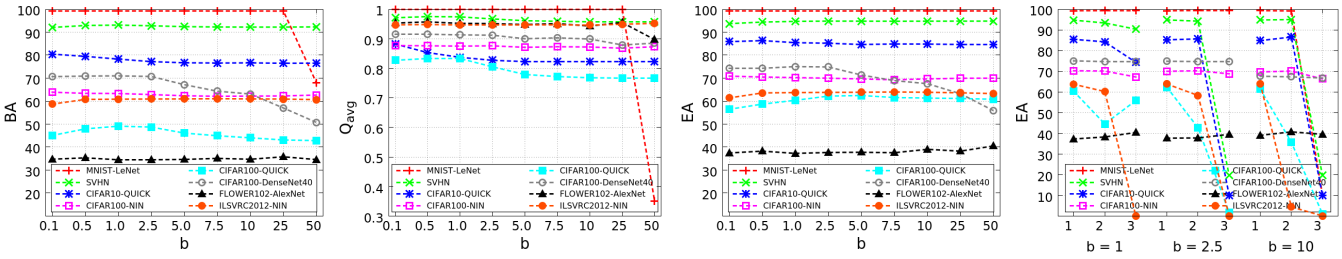
For the EA, we see in Fig. 4 (c) that b = 2.5 generally exhibits the highest EAs. Moreover, the EAs of b = 2.5 are more stable than the other values for b. The above observations are probably due to the trade-off between a high BA and a high base model diversity.

D. Effects of reinitializing more layers

In this section, we investigate how reinitializing more layers affects the EA of our method. Reinitializing more layers injects more randomness into the network, which might help generate more diverse base models. As a result, it might increase the EA.

We tested reinitializing a) only the last layer (as proposed in our method), b) the last two layers and c) the last three layers. The idea is that if reinitializing more layers is beneficial, then b) and c) are very likely to exhibit higher EAs than a). We tested three typical values for b, i.e., b = 1, 2.5 and 10.

The results are shown in Fig. 4 (d). We see that the EAs tend to decrease significantly as more layers are reinitialized. Note that the EAs of ILSVRC2012-NIN are 0.10% (i.e., $1/N_C$) for n = 3 with whatever b = 1, 2.5 or 10. By checking the trained DNNs of which EAs are significantly low, we found that they tend to predict a fixed class label. For instance, one DNN base model predicts class label 9 while another DNN base model



(a) BA vs. b (b) Q_{avg} vs. b (c) EA vs. b (d) EA vs. reinitializing n layers

Fig. 4. Left three plots (a), (b), (c): average base model accuracies (BAs), Q statistic (Q_{avg}) and ensemble accuracies (EAs) of our method with respect to different values for b. Right one plot (d): EAs of our method with respect to reinitializing the last n layers. Here "1", "2" and "3" in the x-axis of the plot mean n = 1, n = 2 and n = 3, respectively. We tested the following (N_R, N_M) pairs for all the four plots: MNIST-LeNet (4, 60), SVHN (8, 60), CIFAR10-QUICK (8, 100), CIFAR100-NIN (8, 10), CIFAR100-QUICK (8, 60), CIFAR100-DenseNet40 (1, 5), FLOWER102-AlexNet (3, 10) and ILSVRC2012-NIN (1, 5). The max_iter in retraining for ILSVRC2012-NIN is set to 75,000, which is smaller than 250,000 (as used in Section V-A), due to the long training time.

predicts class label 7 for all the test data. These predictions suggest that the training of DNNs diverges during retraining.

One possible reason of the divergences is that reinitializing more layers requires more *cooperation* between the reinitialized layers and the layers before them in order to ensure that the forward prorogation makes correct predictions. Here cooperation represents appropriate weight updates for reflecting the connectivity between a reinitialized layer and the layers before it. Thus reinitializing more layers increases the difficulty of the optimization task for the new round of retraining, which often outweighs the increased base model diversity (one exception is FLOWER102-AlexNet). Therefore, we suggest reinitializing only the last layer.

E. Variant 1: retraining for DNNs that exhibit high TAs

The essential part of the retraining framework is to repeatedly reinitialize the last layer of a converged DNN and retrain the network. Our method has no requirement on how to set the predefined classes for the training data. Thus we can generate the training data by merging the classes into meta-classes in the way of ECOC or N-ary ECOC and then conduct our retraining procedure. The motivation is to take the advantages of ECOC and N-ary ECOC, which are very effective in generating diverse predictions (cf. Section V-A), to tackle the tasks in which DNNs exhibit very high TAs (e.g., $> 95\%$).

To verify the above idea, we conducted experiments on FLOWER102-AlexNet, which exhibited a TA of 100.00%. We combined retraining with N-ary ECOC (N = 3), which achieved the highest EA (45.59%, as shown in Table III) on FLOWER102-AlexNet. We conducted three rounds of retraining (N_R = 3), which are sufficient for DNNs that exhibit very high TAs (cf. Section V-B). Finally, we obtained an EA of 46.47%, which is 7.74% and 0.88% higher than the original retraining and N-ary ECOC (N = 3), respectively. The above experiments confirm the effectiveness of *Variant 1*.

F. Variant 2: retraining for DNNs that are computationally expensive to train

The retraining framework we discussed so far typically requires to initialize and train N_M DNNs with the normal training procedure and then conduct N_R rounds of retraining.

TABLE V
ENSEMBLE ACCURACIES (EAS), AVERAGE BASE MODEL ACCURACIES (BAS) AND THE Q STATISTIC (Q_{avg}) OF OUR METHOD, SNAPSHOT ENSEMBLE (SE) AND HORIZONTAL VOTING (HV).

	Method	CIFAR100-DenseNet40$\times 20$	ILSVRC2012-NIN$\times 10$
EA	Retraining	**74.23**	**62.65**
	SE	73.93	61.53
	HV	70.41	60.62
BA	Retraining	**71.43**	**61.24**
	SE	71.34	60.22
	HV	70.19	60.11
Q_{avg}	Retraining	**0.953**	**0.977**
	SE	0.957	0.979
	HV	0.997	0.993

Suppose the training time for a single DNN using the normal training procedure and the retraining procedure are T_{normal} and $T_{retrain}$, respectively, then the overall training time of our method is $N_M(T_{normal} + N_R T_{retrain})$. It might be too demanding as modern DNNs can take days or weeks to train a single model [10]. To deal with this issue, we propose to use the idea of "train one, get N_M" as snapshot ensemble [10] or horizontal voting [28] does. That is, we train a single DNN and then retrain it to obtain a pool of base models. Then the overall training time is reduced to $T_{normal} + N_R T_{retrain}$.

We tested this idea on CIFAR100-DenseNet40 and ILSVRC2012-NIN, both of which are relatively computationally expensive to train. The results are shown in Table V. We see that our method outperforms snapshot ensemble and horizontal voting in terms of the EA. The high EAs are due to both the high BAs and the high base model diversities.

Snapshot ensemble and horizontal voting typically suffer from low diversity. Snapshot ensemble explores multiple local minima sequentially with the later one is obtained by directly training the previous one. Horizontal voting only explores a small region around a single local minimum. For our retraining method, the reinitialization of W_{n_l-1} drives the parameters of a DNN to a region that is far from the previous local minimum in the parameter space, rather than continuing from the previous local minimum (as snapshot ensemble does). Note that the performance of a converged DNN with reinitialized W_{n_l-1} is roughly at the level of random guessing. As more rounds of retraining are conducted, the parameters are driven farther and farther from the first converged point in the

parameter space.

VI. Conclusions

In this paper, we proposed a new ensemble learning method called "retraining", which uses multiple DNNs trained in multiple rounds, for image classification tasks. Experiments showed that retraining outperformed the state-of-the-art methods in most cases, especially for DNNs that exhibit relatively low TAs (e.g., $< 95\%$). We also provided two variants of retraining for two difficult tasks.

Here are some suggestions for using retraining. 1) If the training time is not a crucial issue, use the original retraining framework (as proposed in Section III) when the TA is not very high (e.g., $< 95\%$), or combine retraining with ECOC [26] or N-ary ECOC [27] when the TA is very high (e.g., $> 95\%$). 2) If the training time is a crucial issue, train a few DNNs (if possible) and then retrain them, or train a single DNN and then retrain it, to obtain a pool of base models.

ECOC typically requires too many base models to achieve its optimal performance, while N-ary ECOC typically requires to carefully evaluate different values for N. We will address how to effectively combine retraining with ECOC or N-ary ECOC.

Acknowledgment

A part of this research was supported by Grant-in-Aid for Scientific Research JP18H03290 from the Japan Society for the Promotion of Science (JSPS).

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet Classification with Deep Convolutional Neural Networks," in *Proc. NIPS*, 2012, pp. 1097–1105.

[3] J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, Q. V. Le, and A. Y. Ng, "On Optimization Methods for Deep Learning," in *Proc. ICML*, 2011, pp. 265–272.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, London, 2016.

[5] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, "The Loss Surfaces of Multilayer Networks," in *Proc. AISTATS*, 2015.

[6] K. Kawaguchi, "Deep Learning without Poor Local Minima," in *Proc. NIPS*, 2016, pp. 586–594.

[7] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How Transferable are Features in Deep Neural Networks?" in *Proc. NIPS*, 2014, pp. 3320–3328.

[8] Y. Tang, "Deep Learning Using Linear Support Vector Machines," in *Proc. ICML Workshop*, 2013.

[9] C. Cortes and V. Vapnik, "Support-Vector Networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[10] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger, "Snapshot Ensembles: Train 1, Get M for Free," in *Proc. ICLR*, 2017.

[11] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle *et al.*, "Greedy Layer-Wise Training of Deep Networks," *Advances in Neural Information Processing Systems*, vol. 19, pp. 153–160, 2007.

[12] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent, "The Difficulty of Training Deep Architectures and the Effect of Unsupervised Pre-training," in *Proc. AISTATS*, 2009, pp. 153–160.

[13] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *Proc. ICLR*, 2014.

[14] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition," in *Proc. CVPR Workshop*, 2014, pp. 806–813.

[15] Y. Bengio, "Practical Recommendations for Gradient-Based Training of Deep Architectures," in *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 437–478.

[16] B. Singh, S. De, Y. Zhang, T. Goldstein, and G. Taylor, "Layer-Specific Adaptive Learning Rates for Deep Networks," in *Proc. ICMLA*. IEEE, 2015, pp. 364–368.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," in *Proc. IEEE*, 1998, pp. 2278–2324.

[18] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading Digits in Natural Images with Unsupervised Feature Learning," in *Proc. NIPS Workshop*, 2011.

[19] A. Krizhevsky and G. Hinton, "Learning Multiple Layers of Features from Tiny Images," Master's Thesis, Department of Computer Science, University of Toronto, Canada, 2009.

[20] M.-E. Nilsback and A. Zisserman, "Automated Flower Classification over a Large Number of Classes," in *Proc. ICVGIP*, 2008, pp. 722–729.

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[22] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," in *Proc. ACM Multimedia*, 2014, pp. 675–678.

[23] X.-S. Wei, B.-B. Gao, and J. Wu, "Deep Spatial Pyramid Ensemble for Cultural Event Recognition," in *Proc. ICCV Workshop*, 2015, pp. 38–44.

[24] L. Breiman, "Bagging Predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[25] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-line Learning and an Application to Boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[26] T. G. Dietterich and G. Bakiri, "Solving Multiclass Learning Problems via Error-Correcting Output Codes," *Journal of Artificial Intelligence Research*, vol. 2, no. 1, pp. 263–286, 1995.

[27] J. T. Zhou, I. W. Tsang, S.-S. Ho, and K.-R. Muller, "N-ary Error Correcting Coding Scheme," *arXiv preprint arXiv:1603.05850*, 2016.

[28] J. Xie, B. Xu, and Z. Chuang, "Horizontal and Vertical Ensemble with Deep Representation for Classification," in *Proc. ICML Workshop*, 2013.

[29] S. Lee, S. Purushwalkam, M. Cogswell, D. Crandall, and D. Batra, "Why M Heads are Better than One: Training a Diverse Ensemble of Deep Networks," *arXiv preprint arXiv:1511.06314*, 2015.

[30] M. Opitz, H. Possegger, and H. Bischof, "Efficient Model Averaging for Deep Neural Networks," in *Proc. ACCV*, 2016.

[31] E. L. Allwein, R. E. Schapire, and Y. Singer, "Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers," *Journal of Machine Learning Research*, vol. 1, pp. 113–141, 2000.

[32] T. Hastie, S. Rosset, J. Zhu, and H. Zou, "Multi-class Adaboost," *Statistics and its Interface*, vol. 2, no. 3, pp. 349–360, 2009.

[33] G. U. Yule, "On the Association of Attributes in Statistics: With Illustrations from the Material of the Childhood Society," *Philosophical Transactions of the Royal Society of London, Series A*, vol. 194, pp. 257–319, 1900.