Diverse Ensemble Evolution: Curriculum Data-Model Marriage

Tianyi Zhou, Shengjie Wang, Jeff A. Bilmes
Depts. of Computer Science and Engineering, and Electrical and Computer Engineering
University of Washington, Seattle
{tianyizh, wangsj, bilmes}@uw.edu

Abstract

We study a new method ("Diverse Ensemble Evolution (DivE^2)") to train an ensemble of machine learning models that assigns data to models at each training epoch based on each model's current expertise and an intra- and inter-model diversity reward. DivE^2 schedules, over the course of training epochs, the relative importance of these characteristics; it starts by selecting easy samples for each model, and then gradually adjusts towards the models having specialized and complementary expertise on subsets of the training data, thereby encouraging high accuracy of the ensemble. We utilize an intra-model diversity term on data assigned to each model, and an inter-model diversity term on data assigned to pairs of models, to penalize both within-model and cross-model redundancy. We formulate the data-model marriage problem as a generalized bipartite matching, represented as submodular maximization subject to two matroid constraints. DivE^2 solves a sequence of continuous-combinatorial optimizations with slowly varying objectives and constraints. The combinatorial part handles the data-model marriage while the continuous part updates model parameters based on the assignments. In experiments, DivE^2 outperforms other ensemble training methods under a variety of model aggregation techniques, while also maintaining competitive efficiency.

1 Introduction

Ensemble methods [7, 57, 31, 8] are simple and powerful machine learning approaches to obtain improved performance by aggregating predictions (e.g., majority voting or weighted averaging) over multiple models. Over the past few decades, they have been widely applied, consistently yielding good results. For neural networks (NN) in particular, ensemble methods have shown their utility from the early 1980s [72, 28, 39, 10] to recent times [50, 27, 66, 20]. State-of-the-art results on many contemporary competitions/benchmarks are achieved via ensembles of deep neural networks (DNNs), e.g., ImageNet [17], SQuAD [55], and the Kaggle competitions (https://www.kaggle.com/). In addition to boosting state-of-the-art performance of collections of large models, ensembles of small and weak models can achieve performance comparable to much larger individual models, and this can be useful when machine resources are limited. Inference over an ensemble of models, moreover, can be easily parallelized even on a distributed machine.

A key reason for the success of ensemble methods is that the diversity among different models can reduce the variance of the combined predictions and improve generalization. Intuitively, diverse models tend to make mistakes on different samples in different ways (e.g., assigning largest probability to different wrong classes), so during majority voting or averaging, those different mistakes cancel each other out and the correct predictions can prevail. As neural networks grow larger in size and intricacy, their variance correspondingly increases, offering opportunity for reduction by a diverse ensemble of such networks.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

Randomization is a widely-used technique to produce diverse ensembles. Classical ensemble methods such as random initialization [17, 63], random forests [31, 8] and Bagging [7, 19], encourage diversity by randomly initializing starting points/subspaces or resampling the training set for different models. Ensemble-like methods for DNNs, e.g., dropout [61] and swapout [59], implicitly train multiple diverse models by randomly dropping hidden units out during the training of a single model. Diversity can also be promoted by sequentially training multiple models to encourage a difference between the current and previously trained models, such as Boosting [57, 23, 50] and snapshot ensembles [32]. Such sequential methods, however, are hard to parallelize and can lead to long training times when applied to neural networks.

Despite the consensus that diversity is essential to ensemble training, there is little work explicitly encouraging and controlling diversity during ensemble model training. Most previous methods encourage diversity only implicitly, and are incapable of adjusting the amount of diversity precisely based on criterion determined during different learning stages, nor do they have an explicit diversity representation. Some methods implicitly encourage diversity during training, but they rely on learning rate scheduling (e.g., snapshot ensembles [32]) or end-to-end training of an additive combination of models (e.g., mixture of experts [33, 34, 58]) to promote diversity, which is hard to control and interpret.

Moreover, many existing ensemble training methods train all models in the ensemble on all samples in the training set by repeatedly iterating through it, so the training cost increases linearly with the number of models and number of samples. In such case, each model might waste much of its time on a large number of redundant or irrelevant samples that have already been learnt, and that might contribute nearly zero-valued gradients. The performance of an ensemble on each sample only depends on whether a subset of models (e.g., half for majority voting) makes a correct prediction, so it should be unnecessary to train each model on every sample.

In this paper, we aim to achieve an ensemble of models using explicitly encouraged diversity and focused expertise, i.e., each model is an expert in a sufficiently large local-region of the data space, and all the models together cover the entire space. We propose an efficient meta-algorithm "diverse ensemble evolution (DivE^2)", that "evolves" the ensemble adaptively by changing over stages both the diversity encouragement and each model's expertise, and this is based on information available during ensemble training. It does this encouraging both intra- and inter-model diversity. Each training stage is formulated as a hybrid continuous-combinatorial optimization. The combinatorial part solves a data-model-marriage assignment via submodular generalized bipartite matchings; the algorithm explicitly controls the diversity of the ensemble and the expertise of each model by assigning different subsets of the training data to different models. The continuous part trains each model's parameters using the assigned subset of data. At each stage, all the models may be updated in parallel after receiving their assigned data.

A similar approach to encourage inter-model diversity was used in[15] but there diversity of different models is achieved by encouraging diverse subsets of features and the goal was to cluster the features into potentially overlapping groups; here we are encouraging diverse subsets of samples to be assigned to models during the training process and we are matching data samples to models.

We apply DivE^2 to four benchmark datasets, and show that it improves over randomization-based ensemble training methods on a variety of approaches to aggregate ensemble models into a single prediction. Moreover, with model selection based ensemble aggregation (defined below), DivE^2 can quickly reach reasonably good ensemble performance after only a few learning stages even though each individual model has poor performance on the entire training set. Furthermore, DivE^2 exhibits competitive efficiency and good of model expertise interpretability, both of which can be important in DNN training.

2 Diverse Ensemble Evolution (DivE^2): Formulation

2.1 Data-Model Marriage

An ensemble of models can make an accurate prediction on a sample without requiring each model making accurate predictions on that sample [41, 26, 42]. Rather, it requires a subset of models to produce accurate predictions, and the remainder may err in different ways. Hence, rather than training each model on the entire training set, we may in theory assign a data subset to each model. Then, each sample is learned by a subset of models, and different models are trained on different subsets thereby avoiding common mistakes across models.

Consider a weighted bipartite graph (see Fig. 1), with the set of $n = |V|$ training samples V on the left side, the set of $m = |U|$ models U on the right side, and edges $E \triangleq \{(v_j, u_i)|v_j \in V, u_i \in U\}$

connecting all sample-model pairs with edge weights defined by the loss $\ell(v_j; w_i)$ of sample $v_j = (x_j, y_j)$ (where x_j is the features and y_j is the label(s)) on model u_i (where model u_i is parameterized by w_i). We wish to marry samples with models by selecting a subset of edges having overall small loss. We can express this as follows:

$$\max_{\{w_i\}_{i=1}^m} \max_{A \subseteq E} \sum_{(v_j, u_i) \in A} (\beta - \ell(v_j; w_i)), \tag{1}$$

where $\beta - \ell(v_j; w_i)$ translates loss to reward (or accuracy), and β is a constant larger than any per-sample loss on any model, i.e., $\beta \geq \ell(v_j; w_i), \forall i, j$ [1].

With no constraints, all edges are selected thus requiring all models to learn all samples. As mentioned above, for ensembles, every sample need only be learned by a few models, and thus, for any sample v, we may wish to limit the number of incident edges selected to be no greater than k. This can be achieved using partition matroid $\mathcal{M}_V = (E, \mathcal{I}_V)$, where $\mathcal{I}_V = (I_1, I_2, \ldots, I_n)$ and $I_i \subseteq E$. $\mathcal{I}_V$ contains all subsets of E where no sample is incident to more than k edges in any subset, i.e. $\mathcal{I}_V = \{A \subseteq E : |A \cap \delta(v)| \leq k, \forall v \in V\}$, where $\delta(v) \subseteq E$ is the set of edges incident to v (likewise for $\delta(u), u \in U$). Therefore, as long as a selected subset $A \subseteq E$ satisfies the constraint ($A \in \mathcal{I}_V$), every sample is selected by at most k models.

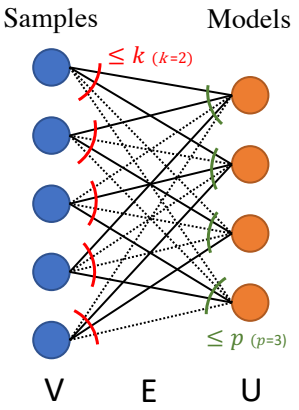

Figure 1: Data-Model Marriage as Bipartite Matching.

With only the constraint $A \in \mathcal{I}_v$, different models can be assigned dramatically differently sized data subsets. In the extremely unbalanced case, k models might get all the training data, while the other models get no data at all. This is obviously undesirable because k models will learn from the same data (no diversity and no specialized and complementary expertise), while the others models learn nothing. Running time also is not improved since training time is linear in the size of the largest assigned data set, which is all of the data in this case. We therefore introduce a second partition matroid constraint $\mathcal{M}_U = (E, \mathcal{I}_U)$, which limits to p the number of samples selected by each model. Specifically, $\mathcal{I}_u = \{A \subseteq E : |A \cap \delta(u)| \leq p, \forall u \in U\}$. Eq. (1) then becomes:

$$\max_{\{w_i\}_{i=1}^m} \max_{A \subseteq E, A \in \mathcal{I}_v \cap \mathcal{I}_u} \sum_{(v_j, u_i) \in A} (\beta - \ell(v_j; w_i)). \tag{2}$$

The interplay between the two constraints (i.e., $\mathcal{I}_v$: k models per sample, and $\mathcal{I}_u$: p samples per model) is important to our later design of a curriculum that leads to a diverse and complementary ensemble. When $mp < nk$, $\mathcal{I}_u$ tends to saturate (i.e., $|A \cap \delta(u)| = p, \forall u \in U$) earlier than $\mathcal{I}_v$. Hence, each model generally has the opportunity to select the top-p easiest samples (i.e., those having the smallest loss) for itself. We call this the "model selecting sample" (or early learning) phase, where each model quickly learns to perform well on a subset of data. On the other hand, when $mp > nk$, $\mathcal{I}_v$ tends to saturate earlier (i.e., $|A \cap \delta(v)| = k, \forall v \in V$), and each sample generally has the opportunity to select the best-k models for itself. We call this the "sample selecting model" (or later learning) phase, where models may develop complementary expertise so that together they can perform accurately over the entire data space. We give conditions on which phase dominates in Lemma 1.

2.2 Inter-model & Intra-model Diversity

To encourage different multiple models to gain different proficiencies, the subsets of training data assigned to different models should be diverse. The two constraints introduced above are helpful to encourage diversity to a certain extent when p and k are small. For example, when $k = 1$ and $p \leq n/m$, no pairs of models share any training sample. Different training samples, however, can still be similar and thus redundant, and in this case the above approach might not encourage diversity when p or k is large. Therefore, we incorporate during training an explicit inter-model diversity term $F_{inter}(A) \triangleq \sum_{i,j \in [m], i<j} F\big((\delta(u_i) \cup \delta(u_j)) \cap A\big)$ and add it to the objective function in Eq. (2). This discourages model pairs from becoming too similar by discouraging their being assigned similar data. The set function $F : 2^E \to \mathbb{R}_+$ is chosen from the large expressive family of submodular functions, which naturally measure the diversity of a set of items [24]. A submodular function satisfies the diminishing return property: given a finite ground set V, any $A \subseteq B \subseteq V$ and an element

[1] Although in theory the loss can be arbitrarily large, in practice, it is usually forced to be upper bounded by a constant for a stable gradient, e.g., a small σ used in $-\log(p_i + \sigma) \leq -\log(\sigma)$ when computing cross entropy loss. Gradient clipping widely used in training neural nets also avoids arbitrarily large loss.

$v \notin B, v \in V$, we have $F(\{v\} \cup A) - F(A) \geq F(\{v\} \cup B) - F(B)$. Submodular functions have been applied to a variety of diversity-driven tasks to achieve good results [45, 44, 3, 54, 25, 22].

Another issue of Eq. (2) is that each model might select easy but redundant samples when constraint $\mathcal{I}_u$ dominates (the model-selecting-sample phase). This is problematic as each model might quickly focus on a small group of easy samples, and may overfit to such small region in the data space. We therefore introduce another set function $F_{intra}(A) = \sum_{i\in[m]} F'(\delta(u_i) \cap A)$ to promote the diversity of samples assigned to each model. Similar to F, we also choose F' to be a submodular function. The optimization procedure now becomes:

$$\max_{W} \max_{A\subseteq E, A\in\mathcal{I}_v\cap\mathcal{I}_u} G(A, W) \triangleq \sum_{(v_j,u_i)\in A} (\beta - \ell(v_j; w_i)) + \gamma F_{inter}(A) + \lambda F_{intra}(A), \quad (3)$$

where γ and λ are two non-negative weights to control the trade-offs between the reward term and the diversity terms, and we denote $W \triangleq \{w_i\}_{i=1}^m$ for simplicity. By optimizing the objective $G(A, W)$, we explicitly encourage model diversity in the ensemble, while ensuring every sample gets learned by k models so that the ensemble can generate correct predictions. A form of this objective has been called "submodular generalized matchings" [1] where it was used to associate peptides and spectra.

3 Diverse Ensemble Evolution (DivE^2): Algorithm

3.1 Solving a Continuous-Combinatorial Optimization

Eq. (3) is a hybrid optimization involving both a continuous variable W and a discrete variable A. It degrades to maximization of a piecewise continuous function $H(W) \triangleq \max_{A\subseteq E, A\in\mathcal{I}_v\cap\mathcal{I}_u} G(A, W)$, with each piece defined by a fixed A achieving the maximum of $G(A, W)$ in a local region of W. Suppose that A is fixed, then maximizing $G(A, W)$ (or $H(W)$) consists of to m independent continuous minimization problems, i.e., $\min_{w_i} \sum_{v_j\in V(A\cap\delta(u_i))} \ell(v_j; w_i)$, $\forall i \in [m]$. Here $V(A) \subseteq V$ denotes the samples incident to the set of edges $A \subseteq E$, so $V(A \cap \delta(u_i))$ is the subset of samples assigned to model u_i. When loss $\ell(\cdot; w_i)$ is convex w.r.t. w_i for every i, a global optimal solution to the above continuous minimization can be obtained by various off-the-shelf algorithms. When $\ell(\cdot; w_i)$ is non-convex, e.g., each model is a deep neural networks, there also exist many practical and provable algorithms that can achieve a local optimal solution, say, by backpropagation.

Algorithm 1 SELECTLEARN$(k, p, \lambda, \gamma, \{w_i^0\}_{i=1}^m)$

Input: $\{v_j\}_{j=1}^n, \{l(\cdot; w_i)\}_{i=1}^m, \pi(\cdot; \eta)$
Output: $\{w_i\}_{i=1}^m$
Initialize: $w_i \leftarrow w_i^0 \ \ \forall i \in [m], t = 0$
while *not "converged"* **do**
  $W \leftarrow \{w_i^t\}_{i=1}^m$, define $G(\cdot, W)$ by W;
  $\hat{A} \leftarrow$ SUBMODULARMAX$(G(\cdot, W), k, p)$;
  if $G(\hat{A}, W) > G(A, W)$ **then**
    $A \leftarrow \hat{A}$;
  end if
  for $i \in \{1, \cdots, m\}$ **do**
    $-\nabla\hat{H}(w_i^t) \leftarrow \frac{\partial}{\partial w_i^t} \sum_{v_j\in V(A\cap\delta(u_i))} \ell(v_j; w_i^t)$;
    $w_i^{t+1} \leftarrow w_i^t + \pi\left(\{w_i^\tau, -\nabla\hat{H}(w_i^\tau)\}_{\tau\in[1,t]}; \eta^t\right)$;
  end for
  $t \leftarrow t + 1$;
end while

Suppose we fix W, then maximizing $G(A, W)$ reduces to the data assignment problem (a generalized bipartite matching problem [43], see Appendix [71] Sec. 5.3 for more details), and the optimal A defines one piece of $H(W)$ in the vicinity of W. Finding the optimal assignment is NP-hard since $G(\cdot, W)$ is a submodular function (a weighted sum of a modular and two submodular functions) and we wish to maximize over a feasibility constraint consisting of the intersection of two partition matroids ($\mathcal{I}_v \cap \mathcal{I}_u$). Thanks to submodularity, fast approximate algorithms [51, 48, 49] exist that find a good quality approximate optimal solution. Let $\hat{H}(W)$ denote the piecewise continuous function achieved when the discrete problem is solved approximately using submodular optimization, then we have $\hat{H}(W) \geq \alpha \cdot H(W)$ for every W, where $\alpha \in [0, 1]$ is the approximation factor.

Therefore, solving the max-max problem in Eq. (3) requires interaction between a combinatorial (submodular in specific) optimizer and a continuous (convex or non-convex) optimizer $\pi(\cdot; \eta)$ [2]. We alternate between the two optimizations while keeping the objective $G(A, W)$ non-decreasing.

[2]The optimizer $\pi(\cdot; \eta)$ can be any gradient descent methods, e.g., SGD, momentum methods, Nesterov's accelerated gradient [52], Adagrad [18], Adam [36], etc. Here the first parameter $\cdot$ can include any historical solutions and gradients, and η is a learning rate schedule (i.e., learning rate is η^t for iteration t).

Intuitively, we utilize the discrete optimization to select a better piece of $\hat{H}(W)$, and then apply the continuous optimization to find a better solution on that piece.

Details are given in Algorithm 1. For each iteration, we compute an approximate solution $\hat{A} \subseteq E$ using submodular maximization SUBMODULARMAX (line 6); in lines 7-9 we compare $\hat{A}$ with the old A on $G(\cdot, W)$ and choose the better one; lines 10-13 run an optimizer $\pi(\cdot; \eta)$ to update each model w_i according to its assigned data. Algorithm 1 always generates a non-decreasing (assuming $\pi(\cdot; \eta)$ does the same using, say, a line search) sequence of objective values. With a damped learning rate, only small adjustments get applied to W and $G(\cdot, W)$. Thus, after a certain point the combinatorial part repeatedly selects the same A (and line 7 eventually is always false), so the algorithm then converges as the continuous optimizer converges. [3]

3.2 Theoretical Perspectives

An interesting viewpoint of the max-max problem in Eq. (3) is its analogy to K-means problems [46]. Eq. (3) strictly generalizes the kmeans objective, by setting $\gamma = \lambda = 0, k = 1, p$ to be the number of desired clusters, and the loss to be the distance metric used in K-means (e.g., L2 distance), and the model to be a real valued vector of having the same dimension as x. Since K-means problem is NP-hard, our objective is also NP-hard.

We next analyze conditions for either of the constraints $(\mathcal{I}_v, \mathcal{I}_u)$ introduced in Section 2.1 to saturate. In the two extreme cases, we know that the "sample selecting model" constraint $\mathcal{I}_v$ saturates when $nk \ll mp$ (e.g., $k = 1$ and $p = n$), and the "model selecting sample" constraint $\mathcal{I}_u$ saturates when $nk \gg mp$ (e.g., $k = m$ and $p = 1$). However, it is not clear what exactly happens between them. The following Lemma shows the precise saturation conditions of the two constraints, with proof details in Section 5.1 of Appendix [71].

Lemma 1. *If* SUBMODULARMAX *is greedy algorithm or its variant, the data assignment $\hat{A}$ produced by lines 6-9 in Algorithm 1 fulfills: 1) $\mathcal{I}_v$ saturates, i.e., $|\hat{A} \cap \delta(v)| = k, \forall v \in V$, and $|\hat{A}| = nk$, if $k < {}^{mp+p}/_{n+(p-1)}$; 2) $\mathcal{I}_u$ saturates, i.e., $|\hat{A} \cap \delta(u)| = p, \forall u \in U$, and $|\hat{A}| = mp$, if $k > {}^{mp-p}/_{n-(p-1)}$; 3) when ${}^{mp+p}/_{n+(p-1)} \leq k \leq {}^{mp-p}/_{n-(p-1)}$, we have $|\hat{A}| \geq \min\{(k-1) + (m-k+1)p, (p-1) + (n-p+1)k\}$.*

As stated above, we can think the objective $H(W)$ as a piecewise function, where each piece is associated with a solution to the discrete optimization problem. Since it is NP-hard to optimize the discrete problem, Algorithm 1 optimizes W on $\hat{H}(W)$, which is defined by the SUBMODULARMAX solutions, rather than on $H(W)$. Algorithm 1 has the following properties.

Proposition 1. *Algorithm 1: (1) generates a monotonically non-decreasing sequence of objective values $G(A; W)$ (assuming $\pi(\cdot; \eta)$ does the same) (2) converges to a stationary point on $\hat{H}(W)$; and (3) for any loss $\ell(u, w)$ that is β-strongly convex w.r.t. w, if* SUBMODULARMAX *has approximation factor α, it converges to a local optimum $\hat{W} \in \operatorname{argmax}_{W \in \mathcal{K}} \hat{H}(W)$ (i.e., $\hat{W}$ is optimal in an local area $\mathcal{K}$) such that for any local optimum $W^*_{loc} \in \mathcal{K}$ on the true objective $H(W)$, we have*

$$\hat{H}(\hat{W}) \geq \alpha H(W^*_{loc}) + \frac{\beta}{2} \cdot \min\{(k-1) + (m-k+1)p, (p-1) + (n-p+1)k\} \cdot \|\hat{W} - W^*_{loc}\|_2^2. \quad (4)$$

The proof is in Section 5.2 of Appendix [71]. The result in Eq. (4) implies that in any local area $\mathcal{K}$, if $\hat{W}$ is not close to W^*_{loc} (i.e., $\|\hat{W} - W^*_{\text{loc}}\|^2$ is large), the algorithm can still achieve an objective $\hat{H}(\hat{W})$ close to $H(W^*_{\text{loc}})$, which is a good approximate solution from the perspective of maximizing $G(A, W)$. Section 5.3 of Appendix [71] shows that the approximation factor is $\alpha = {}^1/_{2+\kappa_G}$ for the greedy algorithm, where κ_G is the curvature of $G(\cdot, W)$. When the weights λ and γ are small, κ_G decreases and $G(\cdot, W)$ becomes more modular. Therefore, the approximation ratio α increases and the lower bound in Eq. (4) improves. For general non-convex losses and models (e.g., DNNs), Eq. (4) degenerates to a weaker bound: $\hat{H}(\hat{W}) \geq \alpha H(W^*_{\text{loc}})$.

3.3 Ensemble Evolution: Curricula for Diverse Ensembles with Complementary Expertise

For a model ensemble to produce correct predictions, we require only that every sample be learnt by a few (small k) models. Optimizing Eq. (3) with small k from the beginning, however, might be harmful as the models are randomly initialized, and using the loss of such early stage models for the edge weights and small k could lead to arbitrary samples being associated and subsequently

[3] Convergence is defined as the gradient $\nabla \hat{H}(W)$ w.r.t. W being zero. In practice, we use $\|\nabla \hat{H}(W)\| \leq \epsilon$ for a small ϵ.

locked to models. We would, instead, rather have a larger k and more use of the diversity terms at the beginning. To address this, we design an ensemble curriculum to guide the training process and to gradually approach our ultimate goal.

Algorithm 2 Diverse Ensemble Evolution (DivE^2)

1: **Input:** $\{(x_j, y_j)\}_{j=1}^n, \{w_i^0\}_{i=1}^m, \pi(\cdot;\eta), \mu, \Delta_k, \Delta_p, T$
2: **Output:** $\{w_i^t\}_{i=1}^m$
3: **Initialize:** $k \leq m, p \geq 1$ s.t. $mp \leq nk$,
$\lambda \in [0,1], \gamma \in [0,1]$
4: **for** $t \in \{1, \cdots, T\}$ **do**
5: $\{w_i^t\}_{i=1}^m \leftarrow$ SelectLearn$(k, p, \lambda, \gamma, \{w_i^{t-1}\}_{i=1}^m)$;
6: $\lambda \leftarrow (1-\mu)\cdot\lambda, \gamma \leftarrow (1-\mu)\cdot\gamma$;
7: $k \leftarrow \max\{\lceil k - \Delta_k \rceil, 1\}, p \leftarrow \min\{\lfloor p + \Delta_p \rfloor, n\}$;
8: **end for**

In Section 2.1, we discussed two ($mp < nk$ and $mp > nk$) extreme training regimes. In the first regime, there are plenty of samples to go around but models may only choose a limited set of samples, so this encourages different models to improve on samples they are already good at. In the first regime, however, inter-model diversity is important to encourage models to become sufficiently different from each other. Intra-diversity is also important in the first regime, since it discourages models from being trained on entirely redundant data. In the second regime, there are plenty of models to go around but samples may choose only a limited number of models. Each model is then given a set of samples that it is particularly good at, and further training further specialization. Since all samples are assigned models, this leads to complementary proficiencies covering the data space.

These observations suggest we start at the first regime $mp \leq nk$ with small p and large k, and gradually switch to the second regime with $mp \geq nk$ by slowly increasing p and decreasing k. In earlier stages, we also should set the diversity weights λ and γ to be large, and then slowly reduce them as we move towards the second regime. It is worth noting that besides intra-model diversity regularization, increasing p is also helpful to expand the expertise of each model since it encourages each model to select more diverse samples. Decreasing k also helps to encourage inter-model diversity since it allows each sample to be shared by fewer models.

In later stages, the solution of Algorithm 1 becomes more exact. With λ and γ decreasing, according to Lemma 2, the curvature κ_G of $G(\cdot, W)$ approaches 0, the approximation factor $\alpha = 1/2+\kappa_G$ of greedy algorithm increases, and the approximate objective $\hat{H}(W) \geq \alpha H(W)$ becomes closer to the true objective $H(W)$. Moreover, during later stages when the "sample selecting model" constraint ($\mathcal{I}_v$) dominates and λ and γ are almost 0, the inner modular maximization is be exactly solved ($\alpha = 1$) and greedy algorithm degenerates to simple sorting.

The detailed diverse ensemble evolution (DivE^2) procedure is shown in Algorithm 2. The curriculum is composed of T stages. Each stage uses SelectLearn (Algorithm 1) to (approximately) solve a continuous-combinatorial optimization in the form of Eq. (3) with pre-specified values of (k, p, λ, γ) and initialization $\{w_i^{t-1}\}_{i=1}^m$ from the previous episode as a warm start (line 5). The procedure reduces λ and γ by a multiplicative factor $(1-\mu)$ in line 6, linearly decreases k by Δ_k and additively increases p by Δ_p, in Line 7. Both k and p are restricted to be integers and within the legal ranges, i.e., $k \in [1, m]$ and $p \in [1, n]$. The warm start initialization is similar in spirit to continuation schemes used in previous curriculum learning (CL) [6, 5, 4, 35, 2, 60, 70] and SPL [40, 64, 62, 65], to avoid getting trapped in local minima and to stabilize optimization. As consecutive problems have the same form with similar parameters (k, p, λ, γ), the solution to the previous problem might still evaluate well on the next one. Hence, instead of running lines 5-14 in Algorithm 1 until full convergence (as instructed by line 4), we run them for ≤ 10 iterations for reduced running time.

4 Experiments

We apply three different ensemble training methods to train ensembles of neural networks with different structures on four datasets, namely: (1) MobileNetV2 [56] on CIFAR10 [38]; (2) ResNet18 [29] on CIFAR100 [38]; (3) CNNs with two convolutional layers[4] on Fashion-MNIST ("Fashion" in all tables) [69]; (4) and lastly CNNs with six convolutional layers on STL10 [12][5]. The three training methods include DivE^2 and two widely used approaches as baselines, which are

- Bagging(BAG)[7]: sample a new training set of the same size as the original one (with replacement) for each model, and train it for several epochs on the sampled training set.

[4] A variant of LeNet5 with 64 kernels for each convolutional layer.
[5] The network structure is from https://github.com/aaron-xichen/pytorch-playground.

- RandINIT(RND): randomly initialize model weights of each model, and train it for several epochs on the whole training set.

Details can be found in Table 3 of Appendix [71]. We everywhere fix the number of models at $m = 10$, and use ℓ_2 parameter regularization on w with weight 1×10^{-4}. In DivE^2's training phase, we start from $k = 6, p = {}^{n}/_{2m}$ and linearly change to $k = 1, p = {}^{3n}/_{m}$ in $T = 200$ episodes. We employ the "facility location" submodular function [14, 45] for both the intra and inter-model diversity, i.e., $F(A) = \sum_{v' \in V} \max_{v \in V(A)} \omega_{v,v'}$ where $\omega_{v,v'}$ represents the similarity between sample v and v'. We utilize a Gaussian kernel for similarity using neural net features $z(v)$ for each v, i.e., $\omega_{v,v'} = \exp\left(-\|z(v)-z(v')\|^2/2\sigma^2\right)$, where σ is the mean value of all the ${}^{n(n-1)}/_{2}$ pairwise distances. For every dataset, we train a neural networks on a small random subset of training data (e.g., hundreds of samples) for one epoch, and use the inputs to the last fully connected layer as features z. These features are also used in the Top-k DCS-KNN approach (below) to compute the pairwise ℓ_2 distances to find the K nearest neighbors.

4.1 Aggregation Methods using an Ensemble of Models

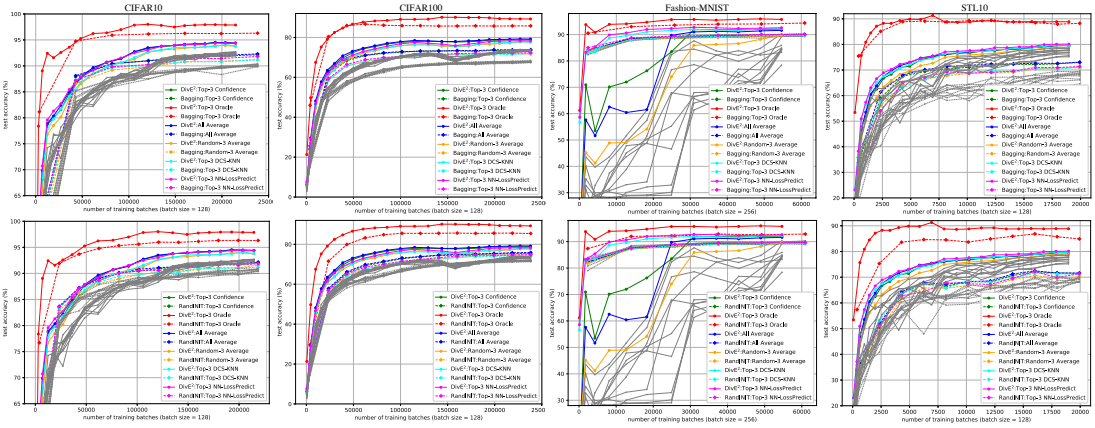

Figure 2: Compare DivE^2 with Bagging(upper row) and RandINIT(lower row) in terms of test accuracy (%) vs. number of training batches on CIFAR10, CIFAR100, Fashion-MNIST and STL10, with $m = 10$ and $k = 3$.

For ensemble model aggregation, when applying a trained ensemble of models to new samples, we must determine (1) which models to use, and (2) how to aggregate their outputs. Here we mainly discuss the first point about different model selection methods, because the aggregation we employ is either an evenly or a weighted average of the selected model outputs. Static model selection methods [72, 10, 53] choose a subset of models from the ensemble and apply it to all samples. By contrast, dynamic classifier selection (DCS) [11, 47, 73, 16] selects different subsets of models to be aggregated for each sample. KNN based DCS [68, 37] is a widely used method that usually achieves better performance than other DCS and static methods. When training, DivE^2 assigns different subsets of samples to different models, so for aggregation, we may benefit more from using sample-specific model selection methods. Therefore, we focus on DCS-type methods, in particular, the following:

- Top-k Oracle: average the outputs (e.g., logits before applying softmax) of the top-k models with the smallest loss on the given sample. It requires knowing the true label, and thus is a cheating method that cannot be applied in practice. However, it shows a useful upper bound on the other methods that select k models for aggregation.
- All Average: evenly average the outputs of all m models.
- Random-k Average: randomly select k models and average their outputs.
- Top-k Confidence: select the top-k models with the highest confidence (i.e., highest probability of the predicted class) on the given sample, and average their outputs.
- Top-k DCS-KNN: apply an KNN based DCS method, i.e., find the K nearest neighbors of the given sample from the training data, select the top-k models assigned to the K nearest neighbors by Top-k Oracle, and average their outputs.
- Top-k NN-LossPredict: train an L2-regression neural nets with m outputs to predict the per-sample losses on the m models by using a training set composed of all training samples and their losses on

the trained models. For aggregation, select the top-k models with the smallest predicted losses on the given sample, and average their outputs.

We compare the three training methods used with the aforementioned aggregation methods with different k[6]. We summarize the highest test-set accuracy when $k = 3$ in Table 2, and show how the test accuracy improves as training proceeds (i.e., as the total training batches on all models increases) in Fig. 2. In Fig. 2, solid curves denote DivE^2, while dashed curves denote the three baseline training methods. Different colors refer to different aggregation methods, and gray curves represent single model performance (gray solid curves denote models trained by DivE^2, while gray dashed curves denote models trained by other baselines). Similar results for $k = 5$ and $k = 7$ can be found in Appendix [71]. In addition, we also tested DivE^2 without the "model selecting sample" constraint and any diversity, which equals to [41, 26, 42] in multi-class case. It achieves a test accuracy of 90.11% (vs. 94.36% of DivE^2) on CIFAR10 and 71.01% (vs. 78.89% of DivE^2) on CIFAR100 when using Top-3 NN-LP for aggregation.

Table 1: Total time (secs.) of DivE^2 and time only on SUBMODULARMAX.

Dataset	CIFAR10	CIFAR100	Fashion	STL10
Total time	26790.75s	34658.27s	2922.89s	4065.81s
SUBMODULARMAX	1857.36s	2697.36s	81.64s	378.84s

Top-k Oracle (cheating) is always the best, and provides an upper bound. In addition, DivE^2 usually has higher upper bound than others, and thus has more potential for future improvement. Solid curves (DivE^2) are usually higher than dashed curves (other baselines) in later stages, no matter which aggregation method is used. Although diversity introduces more difficult samples and lead to slower convergence in early stages, it helps accelerate convergence in later stages. Although the test accuracy on single models achieved by DivE^2 is usually lower than those obtained by other baselines, the test accuracy on the ensemble is better. This indicates that different models indeed develop different local expertise. Hence, each model performs well good only in a local region but poorly elsewhere. However, their expertise is complementary, so the overall performance of the ensemble outperforms other baselines. We visualize the expertise of each model across different classes in Fig. 3 of Appendix [71] for Fashion-MNIST as an example. Among all aggregation methods, Top-k NN-LossPredict and Top-k DCS-KNN show comparable or better performance than other aggregation methods, but require much less aggregation costs when k is small. As shown in Appendix [71], when changing k from minority ($k = 3$) to majority ($k = 7$), the test accuracy of these two aggregation methods usually improves by a large margin. According to Table 1, DivE^2 only requires a few extra computational time for data assignment. The model training dominates the computations but is highly parallelizable since the updates on different models are independent.

Table 2: The highest test accuracy (%) achieved by different combinations of ensemble training and aggregation methods on four datasets, with $k = 3$. DivE^2 usually requires less training time than others to achieve the highest accuracy. The best non-cheating test accuracy (i.e., not Top-k Oracle) is highlighted below.

Train:Aggregation	CIFAR10	CIFAR100	Fashion	STL10
BAG:Top-k Oracle (Cheat)	97.85	88.02	95.60	89.13
BAG:All Average	93.69	73.12	91.24	74.96
BAG:Random-k Avg.	93.05	72.86	91.00	74.03
BAG:Top-k Confidence	93.51	74.59	90.81	75.76
BAG:Top-k DCS-KNN	92.86	73.06	91.39	74.07
BAG:Top-k NN-L.P.	93.45	73.62	92.38	75.16
RND:Top-k Oracle (Cheat)	97.80	87.01	95.71	89.54
RND:All Average	93.28	75.71	91.13	77.13
RND:Random-k Avg.	93.11	75.56	90.77	76.75
RND:Top-k Confidence	93.51	76.54	91.07	77.93
RND:Top-k DCS-KNN	93.18	75.72	92.01	77.23
RND:Top-k NN-L.P.	93.69	76.69	92.48	77.28
DivE^2:Top-k Oracle (Cheat)	98.01	90.12	96.40	90.18
DivE^2:All Average	94.20	**79.12**	86.16	78.95
DivE^2:Random-k Avg.	93.26	77.69	82.75	78.59
DivE^2:Top-k Confidence	94.05	78.76	92.10	79.38
DivE^2:Top-k DCS-KNN	93.81	77.61	92.10	79.23
DivE^2:Top-k NN-L.P.	**94.36**	78.89	**92.76**	**80.49**

Acknowledgments This material is based upon work supported by the National Science Foundation under Grant No. IIS-1162606, the National Institutes of Health under award R01GM103544, and by a Google, a Microsoft, and an Intel research award. This research is also supported by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

[6]The k used in aggregation fixed, and is different from the k in training (which decreases from 6 to 1).

References

[1] Wenruo Bai, Jeffrey Bilmes, and William S. Noble. Bipartite matching generalizations for peptide identification in tandem mass spectrometry. In *7th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics (ACM BCB)*, ACM SIGBio, Seattle, WA, October 2016. ACM, ACM SIGBio.

[2] Sumit Basu and Janara Christensen. Teaching classification boundaries to humans. In *AAAI*, pages 109–115, 2013.

[3] Dhruv Batra, Payman Yadollahpour, Abner Guzman-Rivera, and Gregory Shakhnarovich. Diverse m-best solutions in markov random fields. In *ECCV*, pages 1–16, 2012.

[4] Yoshua Bengio. *Evolving Culture Versus Local Minima*, pages 109–138. Springer Berlin Heidelberg, 2014.

[5] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[6] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML*, pages 41–48, 2009.

[7] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

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

[9] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 535–541, 2006.

[10] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, 2004.

[11] Paulo R. Cavalin, Robert Sabourin, and Ching Y. Suen. Dynamic selection approaches for multiple classifier systems. *Neural Computing and Applications*, 22(3):673–688, 2013.

[12] Adam Coates, Honglak Lee, and Andrew Y. Ng. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*, pages 215–223, 2011.

[13] M. Conforti and G. Cornuejols. Submodular set functions, matroids and the greedy algorithm: tight worst-case bounds and some generalizations of the Rado-Edmonds theorem. *Discrete Applied Mathematics*, 7(3):251–274, 1984.

[14] G. Cornuéjols, M. Fisher, and G.L. Nemhauser. On the uncapacitated location problem. *Annals of Discrete Mathematics*, 1:163–177, 1977.

[15] Andrew Cotter, Mahdi Milani Fard, Seungil You, Maya Gupta, and Jeff Bilmes. Constrained interacting submodular groupings. In *International Conference on Machine Learning (ICML)*, Stockholm, Sweden, July 2018.

[16] Rafael M.O. Cruz, Robert Sabourin, and George D.C. Cavalcanti. Dynamic classifier selection: Recent advances and perspectives. *Information Fusion*, 41:195–216, 2018.

[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[18] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[19] B. Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26, 1979.

[20] Gamaleldin F. Elsayed, Shreya Shankar, Brian Cheung, Nicolas Papernot, Alex Kurakin, Ian J. Goodfellow, and Jascha Sohl-Dickstein. Adversarial examples that fool both human and computer vision. *arXiv*, 2018.

[21] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. An analysis of approximations for maximizing submodular set functions-II. *Mathematical Programming Studies*, 8, 1978.

[22] Madalina Fiterau and Artur Dubrawski. Projection retrieval for classification. In *Advances in Neural Information Processing Systems 25*, pages 3023–3031. 2012.

[23] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[24] Satoru Fujishige. *Submodular functions and optimization*. Annals of discrete mathematics. Elsevier, 2005.

[25] Jennifer Gillenwater, Alex Kulesza, and Ben Taskar. Near-optimal map inference for determinantal point processes. In *NIPS*, pages 2735–2743, 2012.

[26] Abner Guzmán-rivera, Dhruv Batra, and Pushmeet Kohli. Multiple choice learning: Learning to produce multiple structured outputs. In *Advances in Neural Information Processing Systems 25*, pages 1799–1807. 2012.

[27] Shizhong Han, Zibo Meng, AHMED-SHEHAB KHAN, and Yan Tong. Incremental boosting convolutional neural network for facial action unit recognition. In *Advances in Neural Information Processing Systems (NIPS)*, pages 109–117. 2016.

[28] L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[30] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[31] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282, 1995.

[32] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. Snapshot ensembles: Train 1, get M for free. In *International Conference on Learning Representations (ICLR)*, 2017.

[33] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computing*, 3(1):79–87, 1991.

[34] Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural Computing*, 6(2):181–214, 1994.

[35] Faisal Khan, Xiaojin (Jerry) Zhu, and Bilge Mutlu. How do humans teach: On curriculum learning and teaching dimension. In *NIPS*, pages 1449–1457, 2011.

[36] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[37] Albert H. R. Ko, Robert Sabourin, and Alceu Souza Britto, Jr. From dynamic classifier selection to dynamic ensemble selection. *Pattern Recognition*, 41(5):1718–1731, 2008.

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

[39] Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 231–238. 1995.

[40] M. Pawan Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In *NIPS*, pages 1189–1197, 2010.

[41] Stefan Lee, Senthil Purushwalkam, Michael Cogswell, David J. Crandall, and Dhruv Batra. Why m heads are better than one: Training a diverse ensemble of deep networks. *arXiv*, abs/1511.06314, 2015.

[42] Stefan Lee, Senthil Purushwalkam Shiva Prakash, Michael Cogswell, Viresh Ranjan, David Crandall, and Dhruv Batra. Stochastic multiple choice learning for training diverse deep ensembles. In *Advances in Neural Information Processing Systems 29*, pages 2119–2127. 2016.

[43] Hui Lin and Jeff Bilmes. Word alignment via submodular maximization over matroids. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 170–175. Association for Computational Linguistics, 2011.

[44] Hui Lin and Jeff A. Bilmes. A class of submodular functions for document summarization. In *ACL*, pages 510–520, 2011.

[45] Hui Lin, Jeff A. Bilmes, and Shasha Xie. Graph-based submodular selection for extractive summarization. In *Proc. IEEE Automatic Speech Recognition and Understanding (ASRU)*, Merano, Italy, December 2009.

[46] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory (TIT)*, 28(2):129–137, 1982.

[47] Christopher J. Merz. *Dynamical Selection of Learning Algorithms*, pages 281–290. Springer New York, 1996.

[48] Michel Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques*, volume 7 of *Lecture Notes in Control and Information Sciences*, chapter 27, pages 234–243. Springer Berlin Heidelberg, 1978.

[49] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrák, and Andreas Krause. Lazier than lazy greedy. In *AAAI*, pages 1812–1818, 2015.

[50] Mohammad Moghimi, Mohammad Saberian, Jian Yang, Li-Jia Li, Nuno Vasconcelos, and Serge Belongie. Boosted convolutional neural networks. In *British Machine Vision Conference (BMVC)*, 2016.

[51] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions-I. *Mathematical Programming*, 14(1):265–294, 1978.

[52] Yurii Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1):127–152, 2005.

[53] Ioannis Partalas, Grigorios Tsoumakas, and Ioannis Vlahavas. Focused ensemble selection: A diversity-based method for greedy ensemble selection. In *European Conference on Artificial Intelligence (ECML)*, pages 117–121, 2008.

[54] Adarsh Prasad, Stefanie Jegelka, and Dhruv Batra. Submodular meets structured: Finding diverse subsets in exponentially-large structured item sets. In *NIPS*, pages 2645–2653, 2014.

[55] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. In *EMNLP*, 2016.

[56] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *arXiv*, 2018.

[57] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[58] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017.

[59] Saurabh Singh, Derek Hoiem, and David Forsyth. Swapout: Learning an ensemble of deep architectures. In *Advances in Neural Information Processing Systems (NIPS)*, pages 28–36. 2016.

[60] Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. Baby Steps: How "Less is More" in unsupervised dependency parsing. In *NIPS 2009 Workshop on Grammar Induction, Representation of Language and Language Learning*, 2009.

[61] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 15:1929–1958, 2014.

[62] James Steven Supancic III and Deva Ramanan. Self-paced learning for long-term tracking. In *CVPR*, pages 2379–2386, 2013.

[63] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 00, pages 1–9, 2015.

[64] Kevin Tang, Vignesh Ramanathan, Li Fei-fei, and Daphne Koller. Shifting weights: Adapting object detectors from image to video. In *NIPS*, pages 638–646, 2012.

[65] Ye Tang, Yu-Bin Yang, and Yang Gao. Self-paced dictionary learning for image classification. In *MM*, pages 833–836, 2012.

[66] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel. Ensemble adversarial training: Attacks and defenses. In *International Conference on Learning Representations (ICLR)*, 2018.

[67] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 550–558. 2016.

[68] K. Woods, W. P. Kegelmeyer, and K. Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):405–410, 1997.

[69] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[70] Tianyi Zhou and Jeff Bilmes. Minimax curriculum learning: Machine teaching with desirable difficulties and scheduled diversity. In *International Conference on Learning Representations (ICLR)*, 2018.

[71] Tianyi Zhou, Shengjie Wang, and Jeff Bilmes. Supplementary material for diverse ensemble evolution. In *NIPS*, 2018.

[72] Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. Ensembling neural networks: Many could be better than all. *Artificial Intelligence*, 137(1):239–263, 2002.

[73] Xingquan Zhu, Xindong Wu, and Ying Yang. Dynamic classifier selection for effective mining from noisy data streams. In *Data Mining, 2004. ICDM '04. Fourth IEEE International Conference on*, pages 305–312, 2004.

5 Appendix

We define $\hat{A}_v \triangleq \hat{A} \cap \delta(v)$ (edges incident to sample v), and $\hat{A}^u \triangleq \hat{A} \cap \delta(u)$ (edges incident to model u) for simplicity.

5.1 Proof of Lemma 1

Proof. For the monotone non-decreasing objective $G(\cdot, W)$, greedy algorithm or its variant always tries to add more elements until some constraint(s) is violated. Hence, if the first constraint $\mathcal{I}_v$ does not saturate, i.e., there exists at least one $v' \in V$ such that $|\hat{A}_v| = k - x$ for some integer $1 \leq x \leq m$, and $|\hat{A}^u| = p, \forall u \in U \setminus U(\hat{A}_{v'})$, where $U(\hat{A}_{j'})$ represents the set of models incident to $\hat{A}_{v'}$. That is, for any model u that sample v' is not assigned to, the only reason that it cannot be assigned to sample v' when $|\hat{A}_{v'}| < k$ is that the corresponding second constraint $\mathcal{I}_u$ already saturates. The number of models with saturated constraint $|\hat{A}^u| = p$ is $\left|U \setminus \hat{A}_{v'}\right| = m - k + x$. We then have

$$|\hat{A}| \geq |\hat{A}_{v'}| + \sum_{u \in U \setminus \hat{A}_{v'}} |\hat{A}^u| = (k - x) + (m - k + x)p \geq (k - 1) + (m - k + 1)p. \tag{5}$$

For the other $n - 1$ samples excluding sample v', they need to satisfy the constraint $|\hat{A}_v| \leq k$, and they need to be assigned (with repetition) to the $(m - k + x)$ models such that each model gets p samples, i.e.,

$$(n - 1)k \geq \sum_{v \in V \setminus v'} |\hat{A}_v| \geq \sum_{u \in U \setminus \hat{A}_{v'}} |\hat{A}^u| = (m - k + x)p \geq (m - k + 1)p. \tag{6}$$

Therefore, if the first constraint does not saturate, we must have

$$k \geq \frac{mp + p}{n + (p - 1)}. \tag{7}$$

An equivalent statement due to logical transposition rule is: if $k < {}^{mp+p}/_{n+(p-1)}$, the first constraint must saturate, and $|\hat{A}| = nk$. This completes the proof of the first statement in Lemma 1.

By following similar reasoning, if the second constraint does not saturate, i.e., there exists at least one $u' \in U$ such that $|\hat{A}^{u'}| = p - x$ for certain integer $1 \leq x \leq n$, we have

$$|\hat{A}| \geq |\hat{A}^{u'}| + \sum_{v \in V \setminus \hat{A}^{u'}} |\hat{A}_v| = (p - x) + (n - p + x)k \geq (p - 1) + (n - p + 1)k. \tag{8}$$

For the other $m - 1$ models excluding sample u', they need to satisfy the constraint $|\hat{A}^u| \leq p$, and the $(n - p + x)$ samples need to be assigned (with repetition) to these $m - 1$ models such that each sample is assigned to k models, i.e.,

$$(m - 1)p \geq \sum_{u \in U \setminus u'} |\hat{A}^u| \geq \sum_{v \in V \setminus \hat{A}^{u'}} |\hat{A}_v| = (n - p + x)k \geq (n - p + 1)k. \tag{9}$$

Hence, if the second constraint does not saturate, we must have

$$k \leq \frac{mp - p}{n - (p - 1)}. \tag{10}$$

Similarly, if $k > {}^{mp-p}/_{n-(p-1)}$, the second constraint must saturate, and $|\hat{A}| = mp$. This completes the proof of the second statement in Lemma 1.

By combing the conditions in Eq. (7) and Eq. (10), and the respective lower bounds of $|\hat{A}|$ in Eq. (5) and Eq. (8) under these two conditions, the third statement can be proved. □

5.2 Proof of Proposition 1

Proof. In each iteration, lines 7-9 in Algorithm 1 guarantees that $G(\hat{A}, W^t) \geq G(A, W^t)$ (where the superscript t refers to the iteration index), and the gradient descent on $-\hat{H}(W^t)$ (or equally gradient ascent on $\hat{H}(W^t)$) guarantees that $H(W^{t+1}) \geq H(W^t)$, which implies $G(\hat{A}, W^{t+1}) \geq G(\hat{A}, W^t)$. Hence, we have $G(\hat{A}, W^{t+1}) \geq G(\hat{A}, W^t) \geq G(A, W^t)$, i.e., each iteration of Algorithm 1 does not decrease the objective $G(A, W)$ of the max-max problem in Eq. (3). So the algorithm generates

a monotonically non-decreasing sequence of objective values of $G(A, W)$. This completes the proof of the first statement.

By producing a monotonically non-decreasing sequence of objective values of $G(A, W)$, with a damped learning rate η, Algorithm 1 eventually stays on one piece of $\hat{H}(W)$ and converges to a stationary point on that piece with zero gradient. It will not end by oscillating amongst the non-differentiable boundaries between the pieces on $\hat{H}(W)$ because the algorithm can only visit each boundary point for at most one time due to the monotone non-decreasing objective values. This completes the proof of the second statement.

Given the data assignment $\hat{A}$ produced by lines 6-9 in Algorithm 1, the objective $G(\hat{A}, W)$ can be represented as the sum of $|\hat{A}|$ sample-wise loss functions in the following form.

$$G(\hat{A}, W) = \sum_{u_i \in U} \sum_{v_j \in V(\hat{A}^{u_i})} (\beta - \ell(v_j; w_i)) = \beta|\hat{A}| - \sum_{(v_j, u_i) \in \hat{A}} \ell(v_j; w_i). \tag{11}$$

Because each loss function $\ell(v_j; w_i)$ is β-strongly convex w.r.t. w_i, $-G(\hat{A}, W) = -\hat{H}(W)$ is $\beta|\hat{A}|$-strongly convex, which indicates that for any W^*_{loc} and $\hat{W}$,

$$\hat{H}(\hat{W}) + \nabla\hat{H}(\hat{W})^T(W^*_{\text{loc}} - \hat{W}) - \hat{H}(W^*_{\text{loc}}) \geq \frac{\beta|\hat{A}|}{2}\|W^*_{\text{loc}} - \hat{W}\|_2^2. \tag{12}$$

Since $-\hat{H}(W)$ is $\beta|\hat{A}|$-strongly convex, any stationary point $\hat{W}$ achieved by Algorithm 1 is a local optimal solution within some local area $\mathcal{K}$. Hence, for any local optimal solution $W^*_{\text{loc}} \in \mathcal{K}$ on the true objective $H(W)$, the above inequality in Eq. (12) still holds. In addition, because $\nabla\hat{H}(\hat{W}) = \mathbf{0}$, Eq. (12) becomes

$$\hat{H}(\hat{W}) \geq \hat{H}(W^*_{\text{loc}}) + \frac{\beta|\hat{A}|}{2}\|W^*_{\text{loc}} - \hat{W}\|_2^2. \tag{13}$$

If SUBMODULARMAX has approximation factor α, we further have $\hat{H}(W^*_{\text{loc}}) \geq \alpha \cdot H(W^*_{\text{loc}})$. Substituting this result and the lower bound for $|\hat{S}|$ from Lemma 1 into Eq. (13), we have

$$\hat{H}(\hat{W}) \geq \hat{H}(W^*_{\text{loc}}) + \frac{\beta|\hat{A}|}{2}\|W^*_{\text{loc}} - \hat{W}\|_2^2 \tag{14}$$

$$\geq \alpha H(W^*_{\text{loc}}) + \frac{\beta}{2} \cdot \min\{(k-1) + (m-k+1)p, (p-1) + (n-p+1)k\}\|\hat{W} - W^*_{\text{loc}}\|_2^2. \tag{15}$$

This completes the proof of the third statement. □

5.3 Data Assignment as a Generalized Bipartite Matching Problem

The combinatorial optimization problem in Eq. (3) is a generalized bipartite matching problem [43] with monotone submodular evaluations and two matroid constraints, which is a special case of monotone submodular maximization with p-matroid constraint ($p = 2$). simple greedy algorithm can yield an approximation factor of $\alpha = 1/p+1$ [21]. This result can be further improved when the objective $G(\cdot, W)$ is close to modular. Specifically, α becomes $\alpha = 1/p+\kappa_G$ [13], which depends on the curvature $\kappa_G \in [0, 1]$ defined as

$$\kappa_G \triangleq 1 - \min_{j \in V} \frac{G(j|V\backslash j)}{G(j)}. \tag{16}$$

When $\kappa_G = 0$, $G(\cdot, W)$ is modular, and when $\kappa_G = 1$, $G(\cdot, W)$ is fully curved and the above bound recovers $\alpha = 1/p+1$. The objective $G(\cdot, W)$ in Eq. (3) is weighted sum of a modular function and two submodular functions. It becomes closer to modular as the weights λ and γ for the two submodular functions decrease, and κ_G decreases accordingly. We therefore have the following Lemma:

Lemma 2. *Let $G(A) = M(A) + \lambda F(A)$ where $F(\cdot)$ is a monotone non-decreasing submodular function with curvature κ_F, $M(\cdot)$ is a non-negative modular function, and $\lambda \geq 0$. Then $\kappa_G \leq \frac{\kappa_F}{c/\lambda+1}$ where $c = \min_{j \in V} M(j)/F(j)$.*

Proof. We have

$$\kappa_G = 1 - \min_{j \in V} \frac{M(j) + \lambda F(j|V \setminus j)}{M(j) + \lambda F(j)} = \lambda \cdot \max_{j \in V} \frac{F(j) - F(j|V \setminus j)}{M(j) + \lambda F(j)}$$

$$= \lambda \cdot \max_{j \in V} \frac{1 - \frac{F(j|V \setminus j)}{F(j)}}{\frac{M(j)}{F(j)} + \lambda} \leq \frac{\lambda \cdot \kappa_F}{\min_{j \in V} \frac{M(j)}{F(j)} + \lambda} = \frac{\kappa_F}{c/\lambda + 1}$$

Where $c \triangleq \min_{j \in V} \frac{M(j)}{F(j)}$. □

In this paper, we apply the fast greedy procedure mentioned earlier [51, 48, 49] to the data assignment problem. It secures an approximation factor $\alpha = 1/2+\kappa_G$, but might perform much better in practice.

5.4 Related Work

5.4.1 Three mostly used classical ensemble methods

Bagging [7]: bagging samples different training sets for different models before any training starts, and train all models in parallel, finally average all models' outputs as prediction. It does not adapt with the training process, i.e., the assignment of training data does not depend the performance of any model at any training stage. Multiple models can be trained in parallel so bagging is potentially applicable to deep neural nets, but might perform worse than simple average ensemble or dropout [41].

Boosting [57, 23, 50]: train a sequence of models one after another, and the weight of each training sample to train the next model depends on its classification accuracy achieved on previously trained models. It is adaptive and can build a strong ensemble model from multiple weak learners. However, it is not practical in training deep neural nets that usually require a long training time, because each model needs to wait the previous one to converge. In addition, boosting cannot adaptively adjust the training set during the training process of each model. The data assignment happens before any training begins.

Mixture of Experts (MoE) [33, 34]: they use a gating network to select a subset of models (experts) for each given sample. Because the gating network connects all the models together and forms a modular combination, which is usually a large neural network, end-to-end training is usually required, which is hard to parallelize and might result in expensive computations and heavy memory load. DivE^2 has similar idea of training experts, but is different from MoE methods in that 1) DivE^2 explicitly promotes diverse and complementary expertise on different experts; and 2) DivE^2 does not require end-to-end training (the gating network needs re-training if we remove or add models to the ensemble) and is able to train models in parallel, because each model is independently updated based on the assigned data in each learning stage.

5.4.2 Two mostly used ensemble methods for deep neural nets

Simple average might be the most widely used ensemble methods especially for deep neural nets. It trains different models on the same training set but initialize them randomly (and thereby promote diversity implicitly) at the beginning of optimization. It is simple to use but the diversity cannot be explicitly enforced. Moreover, it trains each model on the whole training set independently, so the training costs increase linearly with the number of models m.

Dropout [61]/Swapout [59]: implicitly gain an ensemble by randomly killing a portion of hidden nodes (i.e., set their outputs to be zeros) or skipping over layers (i.e., layer dropout). They implicitly average multiple models with different structures but with shared weights. They are different from explicitly training multiple models and explicitly enforcing the diversity between them. They can always be combined with other ensemble models including ours to further improve generation performance (in this case each model in the ensemble is implicitly an ensemble of models with different structures), and are orthogonal to methods explicitly training multiple models. ResNet [29] can also been explained as an ensemble of shallow models due to its shortcut link between nonconsecutive layers [67].

5.4.3 Two recently proposed ensemble methods for deep neural nets

Snapshot ensemble [32]: by using a cyclic learning rate scheduling, it can quickly converge to a local minimum, and escape from it by an increasing learning rate and then converge to the next local minimum. By repeating this process for several times, it can achieve multiple local minimum models.

The final ensemble is composed of the last several local minimum models. They can also be easily combined with other ensemble methods. One possible disadvantage of snapshot ensemble is that the computational cost to achieve so many local minimums (note the number could be much larger than the number of models used to compose the ensemble) can be very expensive, because it needs to sequentially get local minimum models one after another.

Sparsely gated mixture of experts [58]: it uses a parameterized gate to combine the outputs of all the models, and the gate is designed to only assign nonzero weights to a small number of models. This has been shown to be effective for some NLP tasks. The sparsity is helpful to develop diverse expertise on different models, but can easily cause imbalance loading problem in practice, as the extremely sparse weights given by the gate may always assign most data to few models. In addition, it needs to train thousands of models together with the gating network as a huge neural net in end-to-end manner, so the computational costs are very expensive, and it is not easy to synchronize the training process and accelerate it in parallel.

5.4.4 Other Related Works

Model compression [9] or knowledge distillation [30] learns a single small model to imitate an ensemble of models, so the aggregation requires much less computation and memory. These methods mainly focus on improving the aggregation efficiency, and can also be applied to the diverse ensemble achieved by DivE^2.

5.5 Discussion

In DivE^2, the parameters k, p, λ, γ defining the learning goal of each stage are gradually change according to a pre-defined schedule. In the future, we plan to use reinforcement learning to train an agent to adaptively select these parameters based some features representing the state of the current learning stage.

In DivE^2, we extend the concept of "machine teaching" to "machine education" by emphasizing the dynamic interaction between teacher (data assignment) and students (models) during the learning process. In machine education, the curriculum is composed of a sequence of learning goals for different learning stages, and each goal (i.e., an optimization in the form of Eq. (3)) is achieved based on the current performance of models and the data distribution. In contrast, machine teaching aims at finding the best "teaching set", which is the final goal of the teacher, but does not delicately optimize the learning process. While machine teaching might be useful to convex optimization, machine education that optimizes the learning process (i.e., the curriculum) is more suitable to non-convex optimization for training deep neural networks.

5.6 More Experiment Details

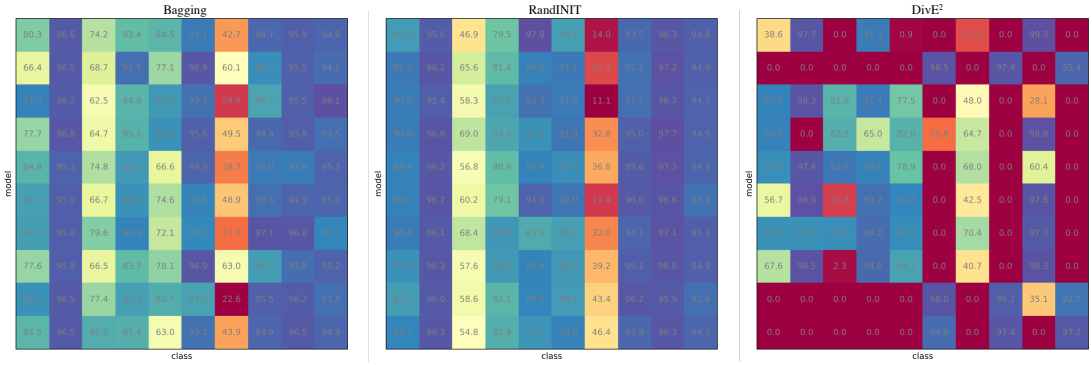

Figure 3: Test accuracy (%) per class on each single model from the ensemble trained by Bagging(left, after 18750 total training batches), RandINIT(middle, after 18750 total training batches) and DivE^2 (right, after 18249 total training batches) on Fashion-MNIST. This figure reflects the expertise of each model on different classes. Comparing to Bagging and RandINIT, the models learned by DivE^2 show diverse and complementary expertise.

Table 3: Details regarding the datasets.

Dataset	CIFAR10	CIFAR100	Fashion	STL10
#Training	50000	50000	60000	5000
#Test	10000	10000	10000	8000
#Feature	$3 \times 32 \times 32$	$3 \times 32 \times 32$	28×28	$3 \times 96 \times 96$
#Class	10	100	10	10

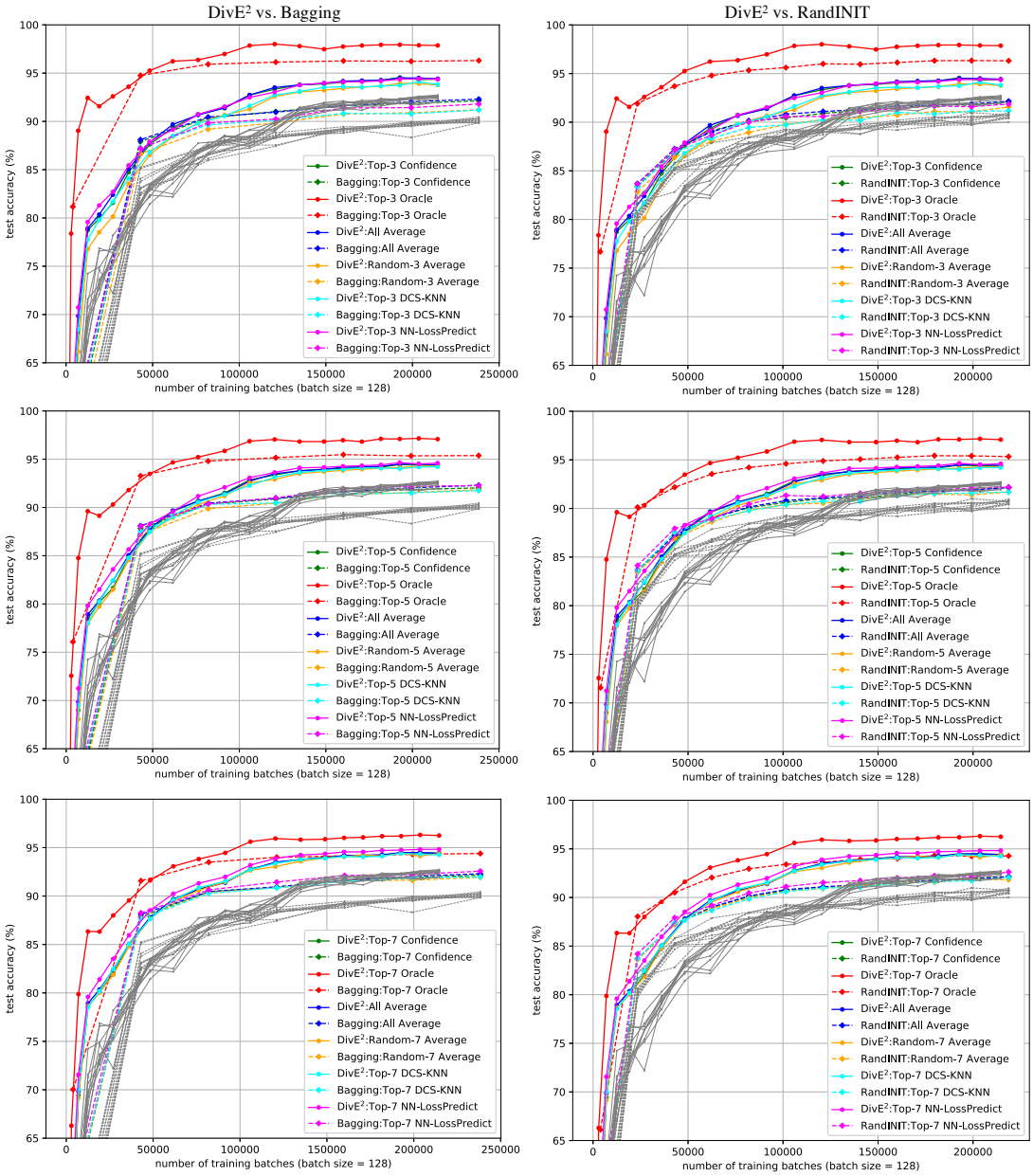

Figure 4: Compare DivE^2 with Bagging(left column) and RandINIT(right column) in terms of test accuracy (%) vs. number of training batches on CIFAR10, with $m = 10$ MobileNetV2 models trained, and using different k values ($k = 3, 5, 7$ from top to bottom) for aggregation.

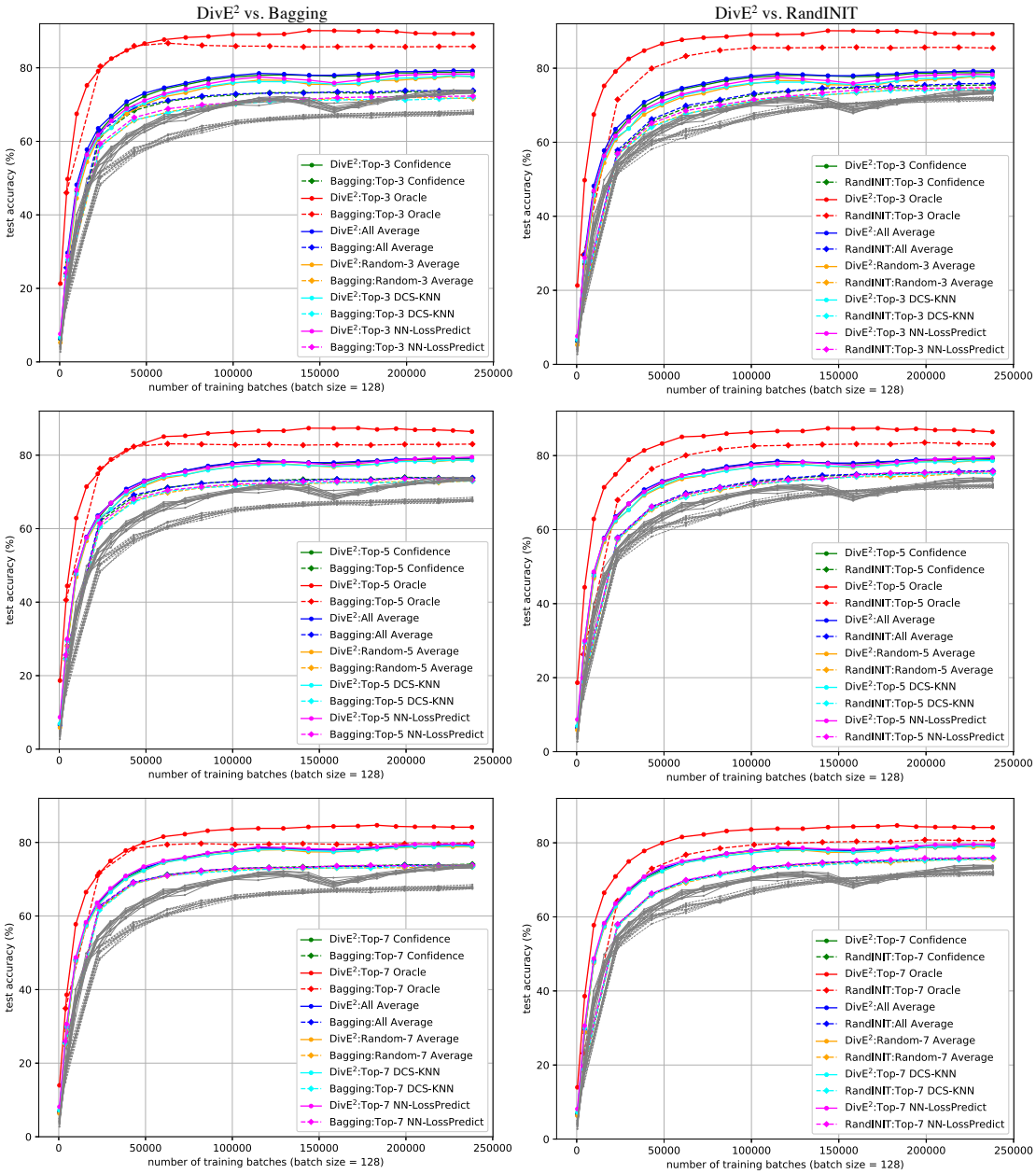

Figure 5: Compare DivE^2 with Bagging(left column) and RandINIT(right column) in terms of test accuracy (%) vs. number of training batches on CIFAR100, with $m = 10$ ResNet18 models trained, and using different k values ($k = 3, 5, 7$ from top to bottom) for aggregation.

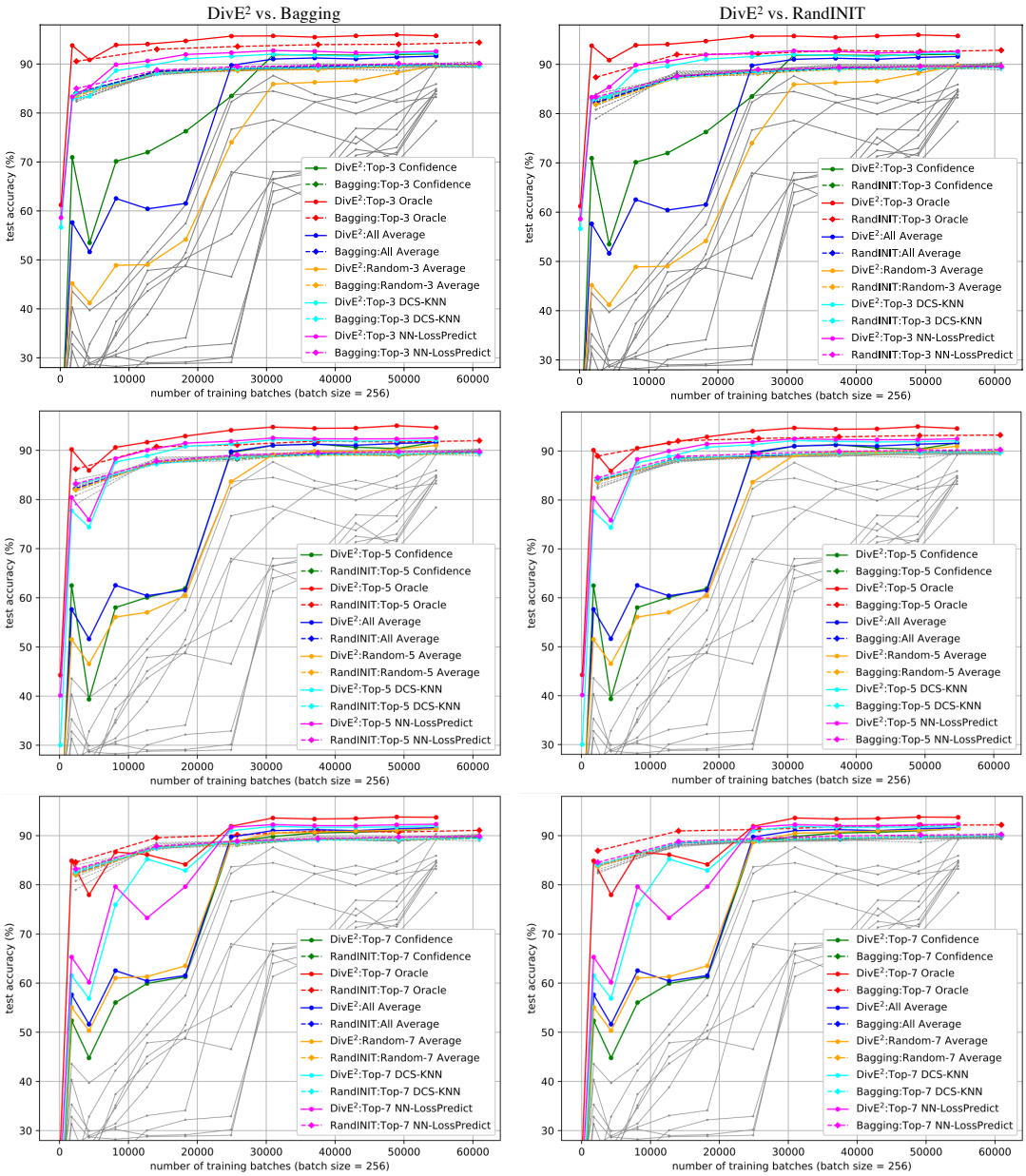

Figure 6: Compare DivE^2 with Bagging(left column) and RandINIT(right column) in terms of test accuracy (%) vs. number of training batches on Fashion-MNIST, with $m = 10$ modified LeNet5 models trained, and using different k values ($k = 3, 5, 7$ from top to bottom) for aggregation.

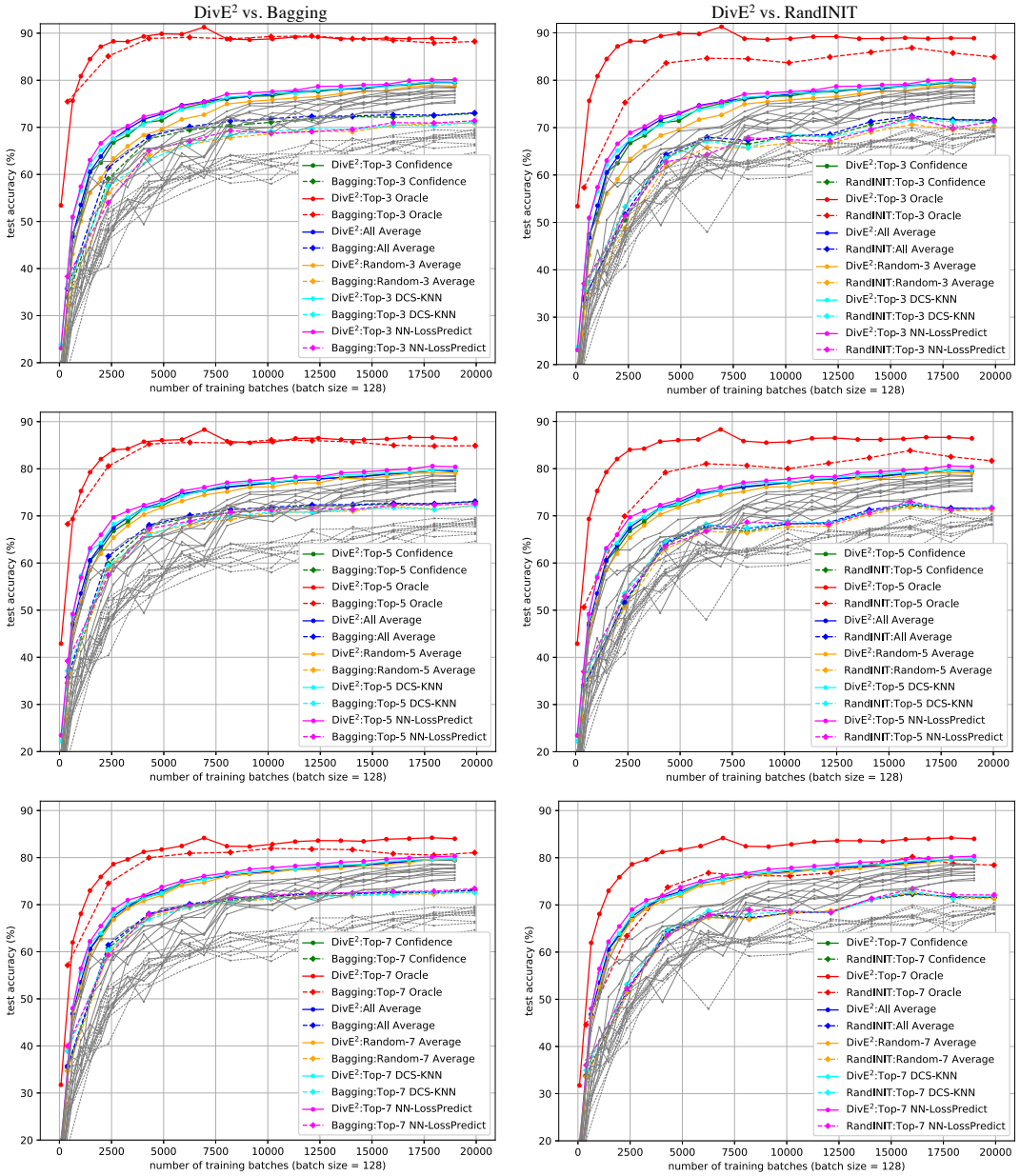

Figure 7: Compare DivE^2 with Bagging(left column) and RandINIT(right column) in terms of test accuracy (%) vs. number of training batches on STL10, with $m = 10$ CNN models trained, and using different k values ($k = 3, 5, 7$ from top to bottom) for aggregation.